Simone Guggiari
ETHZ - HS17

CG

COMPUTER GRAPHICS

SUMMARY

CARLO DEL DON
CSE – VII SEMESTER - ETHZ
11-933-264

Based on the lecture-slides *"Computer Graphics"* and on *"PBRT-Second edition, Matt Pharr, Greg Humphreys"*

SIMONE GUGGIARI
ETH ZÜRICH 2017
LEGI # 14-932-867

Contents

1 Texture Mapping 1
- Texture parametrization . . . 1
- Surface parametrization . . . 1

2 Ray Tracing 2
- Transformations . . . 2
- Light-transport, Assumptions . . . 2
- Forward Ray Tracing . . . 3
- Backward Ray Tracing . . . 3
- Ray-Surface intersection . . . 3

3 Light and Matter 6
- Light emission . . . 6
- Radiometry . . . 6
- Material-Light interaction . . . 8
- Lambertian Reflection . . . 8
- Ideal Specular Reflection . . . 9
- Ideal Specular Refraction . . . 10
- Reflectance VS Refraction . . . 11

4 Appearance Modeling 12
- Microfacet theory . . . 12

5 Monte Carlo 13
- Inversion method . . . 13
- Ambient Occlusion . . . 15
- Improved random sampling . . . 17

6 Direct Illumination 18
- Direct Lighting . . . 19
- Light Sources . . . 21
- Multiple strategies . . . 22
- Environment Lighting . . . 25
- Advanced Camera Models . . . 27

7 Global illumination 28
- Light paths . . . 28
- Rendering equation . . . 28
- Recursive Monte Carlo . . . 29
- Path Tracing . . . 30
- Bidirectional Path Tracing . . . 32

8 Many Lights Rendering 33
- VPL . . . 33
- VSL . . . 34
- Applications . . . 34

9 Photon Mapping 35
- Backward Ray Tracing . . . 35
- Photon Mapping Algorithm . . . 36
- Progressive Photon Mapping . . . 40
- Photon Relaxation . . . 41

10 Acceleration Data Structures 42
- Spatial Subdivision . . . 42
- Bounding Volumes . . . 44

11 Participating Media 46
- Coefficients . . . 46
- Volume Rendering Equation . . . 49
- Phase Function . . . 50
- Analytical Solution . . . 51
- Numerical Solution . . . 53
- Volumetric Path Tracing . . . 54
- Woodcock Tracking . . . 56
- Volumetric Photon Mapping . . . 56
- Photon Beams . . . 58

12 Subsurface Scattering 59
- Diffusion Approximation . . . 59
- Subsurface Scattering Algorithm . . . 61

13 Denoising 62
- Non Local Means Filter . . . 62
- A Posteriori VS A Priori . . . 62

14 Image Based Rendering 63
- Computer Vision . . . 63
- Light Field . . . 64
- Comparison . . . 66

Index 69

1 Texture Mapping

Modeling intricate details using geometry (of the mesh) is difficult and costly. Images can be *pasted* on a mesh by a parametrization.

Texture parametrization

Applying an image on a mesh is the equivalent of finding a parametrization that maps each point $\mathbf{x} = [x, y, z]^T$ in $3D$ space to an image point $\mathbf{u} = [u, v]^T$ that lies in a two dimensional space:

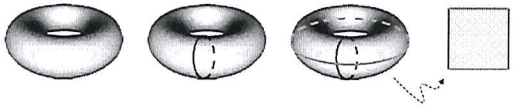
$$P : \mathcal{M} \to [0,1] \times [0,1]$$
$$P\left([x, y, z]^T\right) = [u, v]$$

The texture itself is a function that maps a two dimensional point $\mathbf{u} = [u, v]^T$ to a color:

$$T : [0,1] \times [0,1] \to \text{RGB}$$
$$T\left([u, v]^T\right) = [r, g, b]$$

Surface parametrization

The mapping of each vertex $\mathbf{v}$ to a point $\mathbf{u}_v$ is a hard task for non-analytical surfaces. The mapping of the points lying on the faces of the mesh (i.e. between some vertices) is found exploiting the *barycentric* coordinate system.



Meshes that are **not** isomorphic to a disc (or a square), need to be cut (into smaller pieces), and each of the resulting disc-isomorphic pieces can be mapped to the uv domain. To have a good-looking texture mapping, all those mappings are dependent from each other, as one might want to preserve continuity of the mapping between different pieces.

Harmonic parametrization: Disc-isomorphic meshes can be mapped to the uv domain in a simple way:

Step 1: Map the boundary vertices of the mesh to a circle

Step 2: Find inner vertices mapping by minimizing an harmonic energy:

(i) associate to all edges a spring of a given rest-length having potential energy:

$$E_{ij} = \frac{1}{2} k_{ij} \|\mathbf{v_i} - \mathbf{v_j}\|$$

(ii) find minimum energy state, where all vertices lie on the same $2D$ plane

(iii) choose a spring constant k_{ij} according to one of the following rules:

uniform	**cordal**	**cotan**
$k_{ij} = 1$	$k_{ij} = \frac{1}{\|\mathbf{v_i} - \mathbf{v_j}\|}$	$k_{ij} = \cot \phi_{ij} + \cot \phi_{ji}$

2 Ray Tracing

Transformations

A transformation matrix M has the following structure:

$$M = \begin{bmatrix} m_{11} & m_{21} & m_{31} & m_{41} \\ m_{12} & m_{22} & m_{32} & m_{42} \\ m_{13} & m_{23} & m_{33} & m_{43} \\ m_{14} & m_{24} & m_{34} & m_{44} \end{bmatrix}$$

we can also define special cases of a transformation matrix, i.e. a *translation* and a *scaling* matrix:

$$M_t = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad M_s = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

A point in $3D$ space can be transformed when applied a transformation matrix. Using a $4D$ homogeneous representation of a point $\mathbf{a} = [x, y, z]^T$ we have:

$$\mathbf{a} = \begin{bmatrix} x \\ y \\ z \end{bmatrix} \to \mathbf{a} = \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}$$

Points VS Vectors: The homogeneous representation of points and vectors differ in the value of w:

Point: $w = 1$, **Vector:** $w = 0$

This is useful as a point can be transformed using a translation matrix, while if applied to a vector this should not have any effect. Vectors can be summed together, while an addition of two points makes no sense.

Light-transport, Assumptions

In order to define a model of the physical light transport, we need to make a few assumptions:

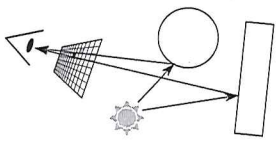
(i) geometrical optics: no diffraction, no polarization, no interference

(ii) light travels in a straight line in a vacuum: no atmospheric scattering or refraction, no gravity effects.

(iii) discrete wavelength of light: RGB

(iv) no non-linear reflecting materials

Forward Ray Tracing

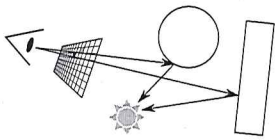


Is the representation of what happens in reality: light (photons) are emitted by a light-source, and are scattered around the scene until (luckily) they hit the camera that registers their radiance.

Even though this is the process that happens in nature, the standard way of performing ray tracing in computer graphics is the *backward raytracing*.

Backward Ray Tracing

This idea of ray tracing is based on the fact, that a lot of rays that are emitted by a light source may never hit the camera.



In order to avoid computational costs to calculate the scattering of those unlucky rays, we can trace rays out of the camera and scatter them into the scene. This can be done (see later) because the surface reflection/refraction *BRDF* is a symmetric function.

Ray-Surface intersection

In the pipeline of the ray tracing algorithm, a lot of ray-surface queries are performed to determine if an object of the virtual scene has been hit by a ray.

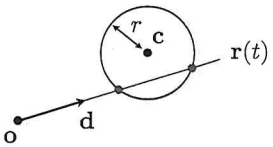
Each ray can be described by its parametric equation:

$$\mathbf{r}(t) = \mathbf{o} + \mathbf{d}t \tag{2.1}$$

where $\mathbf{o}$ is the origin of the ray, $\mathbf{d}$ its direction and t the parameter that *runs* from the origin towards the end of the ray.

Some of the easiest (and most used) surface primitives are the following:

- Sphere
- Plane
- Triangle



Ray-Sphere intersection: The implicit analytical expression for a sphere is the following:

$$\|\mathbf{x} - \mathbf{c}\|^2 - r^2 = 0 \tag{2.2}$$

inserting the ray-equation (2.1) into the parametric equation of the sphere (2.2), we can solve for t finding the parametric distances $t^\star_{1,2}$ from the origin $\mathbf{o}$ at which the ray $\mathbf{r}(t)$ intersects the surface of the sphere.

Note that to find out the solutions $t^\star_{1,2}$ one has to solve a quadratic equation, therefore one of the following situations always happen:

two intersections: the solution of the quadratic equation generates two distinct solutions, the discriminant of the quadratic equation is positive. To find out the nearest intersection point at $\mathbf{r}(t^\star)$ one has to choose the smallest positive solution $t^\star$ between $t^\star_1$ and $t^\star_2$

one intersection: this case never happens in practice, as this corresponds to a ray that is analytically *tangent* to the sphere, the associated discriminant is zero

no intersection: the discriminant of the quadratic equation is negative, the ray does not intersect the sphere

Algebraic solution: inserting the ray-equation into the sphere equation we get:

$$\|\mathbf{o} + \mathbf{d}t - \mathbf{c}\|^2 - r^2 = 0$$

expanding the equation solving the norm-operator $\|\cdot\|^2$ we obtain a quadratic equation:

$$(\mathbf{o}_x + \mathbf{d}_x t - \mathbf{c}_x)^2 + (\mathbf{o}_y + \mathbf{d}_y t - \mathbf{c}_y)^2 + (\mathbf{o}_z + \mathbf{d}_z t - \mathbf{c}_z)^2 - r^2 = 0$$

which reduces to the standard form of a quadratic equation $At^2 + Bt + C = 0$ that can be solved using the *quadratic formula*:

$$t^\star_{12} = \frac{-B \pm \sqrt{B^2 - 4AC}}{2A}$$

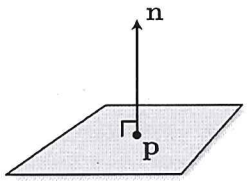
Geometric solution: exploiting *early termination* we can make the ray-sphere intersection query faster.

Ray-Plane intersection: The implicit equation of a plane is given by the general form:

$$ax + by + cz + d = 0 \tag{2.3}$$

where a, b, c are the components of the plane normal vector $\mathbf{n} = [a, b, c]^T$. The implicit equation of the plane is derived by the observation, than any point $\mathbf{x}$ lying on the plane satisfies the following equation:

$$(\mathbf{x} - \mathbf{p}) \cdot \mathbf{n} = 0$$



Substituting the point $\mathbf{x}$ with the ray-equation (2.1) we obtain:

$$(\mathbf{o} + \mathbf{d}t - \mathbf{p}) \cdot \mathbf{n} = 0$$

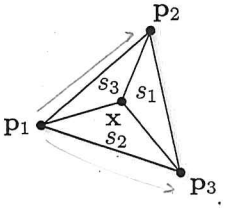
that can be easily solved for t obtaining:

$$t^\star = -\frac{(\mathbf{o} - \mathbf{p}) \cdot \mathbf{n}}{\mathbf{d} \cdot \mathbf{n}}$$

Ray-Triangle intersection: As most meshes are represented as a set of triangles, ray-triangle intersection queries have to be very efficient.

Exploit *barycentric* coordinates inside a triangle:

$$\mathbf{x} = s_1\mathbf{p}_1 + s_2\mathbf{p}_2 + s_3\mathbf{p}_3 \tag{2.4}$$



Inside a triangle, the following condition holds:

$$s_1 + s_2 + s_3 = 1, \quad 0 \le s_i \le 1, i \in 1, 2, 3$$

Solution approach 1: a triangle is nothing but a restricted part of a plane.
To find the intersection point $\mathbf{r}(t^\star)$ with the triangle we could perform a ray-plane intersection test, and verify that the intersection point is inside the triangle:

$$(\mathbf{o} + \mathbf{d}t - \mathbf{p}_1) \cdot \mathbf{n} = 0, \quad \mathbf{n} = \frac{(\mathbf{p}_2 - \mathbf{p}_1) \times (\mathbf{p}_3 - \mathbf{p}_1)}{\|(\mathbf{p}_2 - \mathbf{p}_1) \times (\mathbf{p}_3 - \mathbf{p}_1)\|}$$

Solving for t^* as with the ray-plane intersection gives a point $\mathbf{x}^* = \mathbf{r}(t^*)$ that lies on the same plane as the triangle, we are not guaranteed that the point is indeed inside the triangle. It is possible to check this verifying that $s_i \in [0, 1]$, $i \in 1, 2, 3$ and $s_1 + s_2 + s_3 = 1$.

Solution approach 2: insert the ray-equation (2.1) in the barycentric coordinates (2.4) obtaining:

$$\mathbf{o} + \mathbf{d}t = s_1\mathbf{p_1} + s_2\mathbf{p_2} + s_3\mathbf{p_3}$$

this is a vectorial equation ($3D$ vectors) that can be solved simultaneously for t^*, s_1 and s_2 substituting $s_3 = 1 - s_1 - s_2$. If there is no solution to the vectorial equation, then there is no ray-triangle intersection.

Circle: $C = 2\pi r$, $A = \pi r^2$ $\frac{d}{dr}$

Sphere: $A = 4\pi r^2$, $V = \frac{4}{3}\pi r^3$ $\frac{d}{dr}$

3 Light and Matter

Light emission

Light can be emitted from various bodies, and its generation can be due to different processes.

- Incandescence: Visible light is produced by heat (light bulb).

  Blackbody: is a hypothetical body that completely absorbs thermal radiation incident on it, such bodies do **not** reflect light, therefore they appear black. In a blackbody the emitted light is only due to their temperature, the emitted light spectrum is determined by Plank's law:

  no reflect emit

  check out

$$L_\gamma(T) = \frac{2hc^2}{\lambda^5\left(e^{\frac{hc}{\lambda k_B T}} - 1\right)}$$

  The shape of this function determines the *reddish* color of hot materials, that becomes blue-violet as the material is heated even more.

- Luminescence: light emitted by a substance not resulting from heat, can be caused by chemical reactions, electrical energy,... One example of luminescence is light produced by excited electrons which change their electronic state and emit energy under the form of electromagnetic radiation, i.e. light.

- Fluorescence: occurs when light is absorbed and immediately released by a body at a lower frequency. There exist fluorescent materials like minerals, gases,...

- Phosphorescence: essentially the same as *fluorescence*, except that there might be longer delay between the absorption and emission.

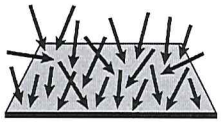
- Chemi- and Bioluminescence: *chemiluminescence* is emission of light as the result of a chemical reaction, *bioluminescence* is *chemiluminescence* that happens inside an organism.

Radiometry

How do we measure light? We can define the following basic quantities (dependent on the wavelength).

- Flux: = Power corresponds to the total energy passing through a surface per unit time:

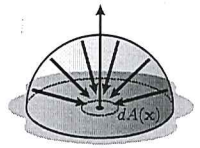
$$\Phi(A), \quad \left[\frac{J}{s} = W\right] \tag{3.1}$$



  Examples: number of photons hitting a surface per unit time, number of photons (leaving) passing through a light-bulb per unit time.

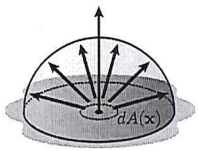
- Irradiance: is defined as the *area density of flux*, i.e. the flux *arriving* per unit area:

$$E(\mathbf{x}) = \frac{\partial\Phi(A)}{\partial A(\mathbf{x})}, \quad \left[\frac{W}{m^2}\right] \tag{3.2}$$



  Example: number of photons hitting a differential patch of a wall per second, divided by the surface of the patch.

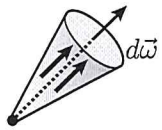
- **Radiosity:** companion of the *irradiance*, defines the *area density of flux, leaving* a surface per unit area:



$$\boxed{B(\mathbf{x}) = \frac{\partial \Phi(A)}{\partial A(\mathbf{x})}, \quad \left[\frac{W}{m^2}\right]} \tag{3.3}$$

Example: number of photons being reflected from a small patch of a wall per second, divided by the surface of the patch.

- **Radiant intensity:** corresponds to the *directional density* of a flux: the power of a flux per *solid angle*:



$$\boxed{I(\vec{\omega}) = \frac{\partial \Phi}{\partial \vec{\omega}}, \quad \left[\frac{W}{sr}\right]} \tag{3.4}$$

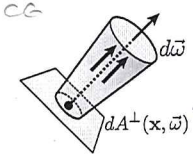
Note that integrating the radiant intensity of a point $\mathbf{x}$ over a sphere enclosing that point, we get the flux:

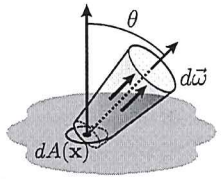
$$\Phi = \int_{S^2} I(\vec{\omega})\, \mathrm{d}\vec{\omega}$$

- **Radiance:** corresponds to the *flux density per solid angle per perpendicular unit area*:

"how much light travels along a ray"
→ most important for CG

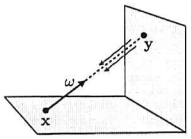
$$\boxed{L(\mathbf{x}, \vec{\omega}) = \frac{\partial I(\vec{\omega})}{\partial A^{\perp}(\mathbf{x}, \vec{\omega})} = \frac{\partial^2 \Phi(A)}{\partial \vec{\omega} \partial A(\mathbf{x}) \cos\theta}, \quad \left[\frac{W}{m^2 sr}\right]} \tag{3.5}$$





The two figures show the two different interpretations of the radiance, one as the radiant intensity $I(\vec{\omega})$ per perpendicular surface area, the other as the flux $\Phi(A)$ per solid angle per perpendicular area.

The radiance is a fundamental quantity for ray tracing: it remains constant along a ray (in vacuum) and has a symmetric property:



$$\boxed{L_i(\mathbf{x}, \vec{\omega}) = L_o(\mathbf{y}, -\vec{\omega})}$$

Relations between radiometric quantities: One can express a radiometric quantity in terms of an other:

Irradiance in terms of Radiance: Integrating the radiance over a hemisphere, we obtain the irradiance:

used BRDF
$$\begin{aligned} L(\mathbf{x}, \vec{\omega}) &= \partial E(\mathbf{x}) / \cos\theta \partial \vec{\omega} \\ \rightarrow L(\mathbf{x}, \vec{\omega}) \cos\theta \partial \vec{\omega} &= \partial E(\mathbf{x}) \\ \int_{H^2} L(\mathbf{x}, \vec{\omega}) \cos\theta \partial \vec{\omega} &= E(\mathbf{x}) \end{aligned}$$

Flux in terms of Radiance: Integrating the radiance over a hemisphere and surface, we obtain the flux:

$$\begin{aligned} E(\mathbf{x}) &= \partial \Phi(A) / \partial A(\mathbf{x}) \\ \int_A E(\mathbf{x})\, \mathrm{d}A &= \Phi(A) \\ \int_A \int_{H^2} L(\mathbf{x}, \vec{\omega}) \cos\theta\, \mathrm{d}\vec{\omega}\, \mathrm{d}A(\mathbf{x}) &= \Phi(A) \end{aligned}$$

Material-Light interaction

We model the behavior of light interacting with a surface reducing the physical interaction that happens in real life to a probability distribution.

The **BRDF** (**B**idirectional **R**eflectance **D**istribution **F**unction) determines how light is reflected/refracted at a given surface depending on its incident angle:

reflected
Sure?
incoming

$$\boxed{f_r(\mathbf{x}, \vec{\omega}_i, \vec{\omega}_r) = \frac{\mathrm{d}L_r(\mathbf{x}, \vec{\omega}_r)}{\mathrm{d}E_i(\mathbf{x}, \vec{\omega}_i)} = \frac{\mathrm{d}L_r(\mathbf{x}, \vec{\omega}_r)}{L_i(\mathbf{x}, \vec{\omega}_i) \cos\theta_i\, \mathrm{d}\vec{\omega}_i}, \quad \left[\frac{1}{sr}\right]} \tag{3.6}$$

where θ_i is the angle between the surface normal and the *incoming* radiance:

$$\theta_i = \arccos(\vec{\omega}_i \cdot \vec{\mathbf{n}})$$

The **BRDF** provides a relation between the incoming radiance L_i and the outgoing L_r. From its definition we can derive the **reflection equation**:

$$\boxed{L_r(\mathbf{x}, \vec{\omega}_r) = \int_{H^2} f_r(\mathbf{x}, \vec{\omega}_i, \vec{\omega}_r) L_i(\mathbf{x}, \vec{\omega}_i) \cos\theta_i\, \mathrm{d}\vec{\omega}_i} \tag{3.7}$$

From the above equation we see that the reflected radiance at a given point $\mathbf{x}$ is given by the integration over all incoming radiances $L_i(\mathbf{x}, \vec{\omega}_i)$ multiplied with the **BRDF** value for the corresponding direction $\vec{\omega}_i$.

BRDF properties: The **BRDF** has to satisfy some physical properties:

Helmholtz reprocipity: the **BRDF** is symmetric, → this allows the equivalence between *forward ray tracing* and *backward ray tracing*:

$$f_r(\mathbf{x}, \vec{\omega}_i, \vec{\omega}_r) = f_r(\mathbf{x}, \vec{\omega}_r, \vec{\omega}_i)$$

Energy conservation: Integrating the **BRDF** over a hemisphere H^2 the following has to hold:

$$\int_{H^2} f_r(\mathbf{x}, \vec{\omega}_i, \vec{\omega}_r) \cos\theta_i\, \mathrm{d}\vec{\omega}_i \leq 1, \quad \forall \vec{\omega}_r$$

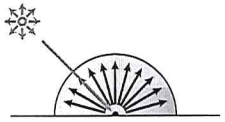
(some might be absorbed)

Isotropic BRDF if the distribution function f_r does *not* change when the material is rotated around its surface normal $\vec{n}$

Anisotropic BRDF if the distribution function f_r has a direct dependency on the orientation of the material.

Lambertian Reflection

also called *ideal diffuse reflection*, the **BRDF** is constant over all solid angles $\mathrm{d}\vec{\omega}_i$: each incoming radiance L_i gives the same contribution (up to the cosine weight) to the reflected radiance L_r.



The constant value of the lambertian **BRDF** can be calculated as follows:

$$\begin{aligned} \boxed{L_r(\mathbf{x}, \vec{\omega}_r)} &= \int_{H^2} f_r(\mathbf{x}, \vec{\omega}_r, \vec{\omega}_i) L_i(\mathbf{x}, \vec{\omega}_i) \cos\theta_i\, \mathrm{d}\vec{\omega}_i \\ L_r(\mathbf{x}) &= f_r \int_{H^2} L_i(\mathbf{x}, \vec{\omega}_i) \cos\theta_i\, \mathrm{d}\vec{\omega}_i \\ &= \boxed{f_r E(\mathbf{x})} \text{ (irradiance)} \end{aligned}$$

As all *incoming* radiance is reflected, the *irradiance* $E(\mathbf{x})$ is equivalent to the radiosity $B(\mathbf{x})$:

$$\begin{aligned} E(\mathbf{x}) &= B(\mathbf{x}) \\ &= \int_{H^2} L_r(\mathbf{x}) \cos\theta_r \, \mathrm{d}\vec{\omega}_r \\ &= L_r(\mathbf{x}) \int_{H^2} \cos\theta_r \, \mathrm{d}\vec{\omega}_r = L_r(\mathbf{x})\pi \end{aligned}$$

Combining the two results we obtain the following:

$$f_r(\mathbf{x}, \vec{\omega}_r, \vec{\omega}_i) = f_r = \frac{1}{\pi} \quad \left(\frac{\rho}{\pi}\right), \rho \text{ is } \textit{albedo}\text{: fraction of reflected radiance} \tag{3.8}$$

For lambertian surfaces, the value of the **BRDF** is constant and the *reflection equation*(3.7) can be simplified to:

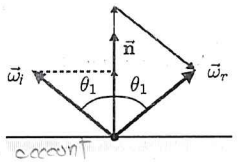
$$L_r(\vec{x}) = \frac{\rho}{\pi} \int_{H^2} L_i(\mathbf{x}, \vec{\omega}_i) \cos\theta_i \, \mathrm{d}\vec{\omega}_i \tag{3.9}$$

where ρ is the *albedo* of the surface ($\rho \in [0, 1]$) which is a linear multiplicative factor to the reflection: if $\rho = 1$, then there is total reflection, no energy loss, if $\rho = 0$ then there is no reflection.

Ideal Specular Reflection

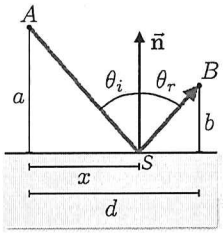
This type of surface allows light to bounce on it and to reflect as a *delta function*: the reflected direction $\vec{\omega}_r$ is determined as follows:

$$\vec{\omega}_r = 2(\vec{\mathbf{n}} \cdot \vec{\omega}_i) \cdot \vec{\mathbf{n}} - \vec{\omega}_i \tag{3.10}$$



The fact that the angle of reflection θ_r is identical to the angle of incidence θ_i ca be derived by *Fermat's principle of reflection*, that states that light follows the path that takes the least time.

Fermat's principle of reflection: given two points $\mathbf{A}$ and $\mathbf{B}$ above a surface, we need to find the shortest path ASB that starts in $\mathbf{A}$, touches the surface at $\mathbf{S}$ and ends in $\mathbf{B}$.



The path length is given by:

$$l = \sqrt{a^2 + x^2} + \sqrt{b^2 + (d - x)^2}$$

varying the distance x we can vary the location of $\mathbf{S}$, and setting the derivative of the length with respect of x to 0 we obtain the optimal distance:

$$\frac{\partial l}{\partial x} = \frac{1}{2}\frac{2x}{\sqrt{a^2 + x^2}} + \frac{1}{2}\frac{2(d - x)(-1)}{\sqrt{b^2 + (d - x)^2}} = 0$$

rearranging the terms of the above equation we obtain:

$$\frac{x}{\sqrt{a^2 + x^2}} = \frac{d - x}{\sqrt{b^2 + (d - x)^2}}$$

It is easy to see that the quantities on both sides of the equation correspond to trigonometric expressions:

$$\sin\theta_i = \sin\theta_r \iff \theta_i = \theta_r \tag{3.11}$$

Ideal Specular Refraction

A lot of materials (specular materials) allow light to *pass through* them, bending its direction. The amount of bending is given by the ratio between the interior and exterior *index of refraction* of the interface.

Given the two indices of refraction it is possible to calculate at which angle θ_r the light continues its path when passing through a specular material being hit with incident angle θ_i. The angle can be calculated with *Snell's law*.

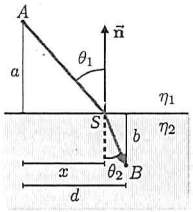
Snell's law Also for this case, according to the fact that light takes always the shortest path to travel from a point $\mathbf{A}$ to $\mathbf{B}$, we can derive the resultant angle at which light is refracted in a material, given the two indices of refraction and the incident angle θ_i.

Index of refraction: the value of the index of refraction is given by the ratio of the speed of light in vacuum and its speed in the material:

$$\eta = \frac{v_{vacuum}}{v_{material}} \tag{3.12}$$

The time it takes to light to travel along the path ASB is given by:

$$t = \frac{\sqrt{a^2 + x^2}}{v_1} + \frac{\sqrt{b^2 + (d - x)^2}}{v_2}$$



setting the derivative of the time with respect to the distance x to 0 we obtain:

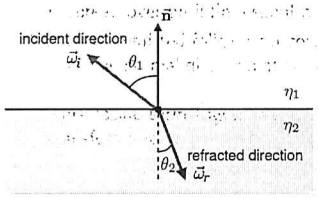
$$\frac{\partial t}{\partial x} = \frac{x}{v_1\sqrt{a^2 + x^2}} - \frac{d - x}{v_2\sqrt{b^2 + (d - x)^2}} = 0$$

rearranging the terms we obtain a trigonometric relation between the angles:

$$\frac{\sin\theta_1}{v_1} = \frac{\sin\theta_2}{v_2} \iff \eta_1 \sin\theta_1 = \eta_2 \sin\theta_2$$

from the above equation we can state *Snell's law*:

$$\frac{\eta_1}{\eta_2} = \frac{\sin\theta_2}{\sin\theta_1} \tag{3.13}$$



From this relation it is possible for any given incident direction $\vec{\omega}_i$, and indices of refraction η_1 and η_2 to determine the refracted outgoing direction $\vec{\omega}_r$:

$$\vec{\omega}_r = -\frac{\eta_1}{\eta_2}\left(\vec{\omega}_i - (\vec{\omega}_i \cdot \vec{\mathbf{n}})\vec{\mathbf{n}}\right) - \vec{\mathbf{n}}\sqrt{1 - \left(\frac{\eta_1}{\eta_2}\right)^2 \left(1 - (\vec{\omega}_i \cdot \vec{\mathbf{n}})^2\right)}$$

Remarks: the index of refraction η depends on the wavelength of the refracted light beam and of its polarization; this dependency gives rise to a number of interesting phenomena like *dispersion* (light being diffracted in a rainbow figure by a prismatic glass), *total internal reflection* (if the incidence angle θ_i is too big (almost 90°) and $\eta_1/\eta_2 \gg 1$, then the refractive surface acts like a reflective one)

Reflectance VS Refraction

There are materials that act as both, reflective and refractive. The question is, how much light is reflected and how much is refracted.

The answer to this question is given by *Fresnel law* that can be derived by *Maxwell's equations*. The amount (fraction) of light that is refracted and reflected can be calculated as follows:

$$\rho_{\parallel} = \frac{\eta_2 \cos\theta_1 - \eta_1 \cos\theta_2}{\eta_2 \cos\theta_1 + \eta_1 \cos\theta_2} \qquad \rho_{\perp} = \frac{\eta_1 \cos\theta_1 - \eta_2 \cos\theta_2}{\eta_1 \cos\theta_1 + \eta_2 \cos\theta_2}$$

$$\text{Reflected: } F_r = \frac{1}{2}\left(\rho_{\parallel}^2 + \rho_{\perp}^2\right) \qquad \text{Refracted: } F_t = 1 - F_r$$

BRDF and BTDF: we can now define the distribution function of a reflective/refractive material.

BRDF for specular reflection:

$$f_r(\mathbf{x}, \vec{\omega}_i, \vec{\omega}_r) = \overbrace{F_r(\vec{\omega}_i)}^{\text{Fresnel reflection}} \frac{\delta\left(\vec{\omega}_r - \overbrace{\mathbf{R}(\vec{\omega}_i, \vec{\mathbf{n}})}^{\text{Reflects } \vec{\omega}_i \text{around } \vec{\mathbf{n}}}\right)}{\cos\theta_i} \tag{3.14}$$

Note that the value of the **BRDF** is non-zero only when the reflected direction $\vec{\omega}_r$ corresponds exactly to the flipped incident direction $\vec{\omega}_i$ The *Fresnel* term F_r is a scalar value that takes into account the fraction of light that is reflected.

BTDF for specular refraction (**B**idirectional **T**ransmissions **D**istribution **F**unction):

$$f_t(\mathbf{x}, \vec{\omega}_r, \vec{\omega}_i) = \left(\frac{\eta_1}{\eta_2}\right)^2 \overbrace{(1 - F_r(\vec{\omega}_i))}^{\text{Fresnel refraction}} \frac{\delta\left(\vec{\omega}_r - \overbrace{\mathbf{T}(\vec{\omega}_i, \vec{\mathbf{n}})}^{\text{Refracts } \vec{\omega}_i \text{around } \vec{\mathbf{n}}}\right)}{\cos\theta_i} \tag{3.15}$$

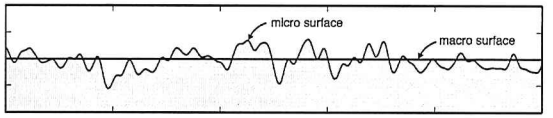
Also here, note that the **BTDF** is non-zero only if the transmitted direction $\vec{\omega}_r$ corresponds to the one provided by *Snell's law*. The *Fresnel* term $1 - F_r$ is a scalar value that takes into account the fraction of light that is refracted.

Remark: note that the cosine term is in the denominator so that it will be canceled in the reflection equation (3.7) as the cosine scaling is already included in the *Fresnel* term F_r.

4 Appearance Modeling

Microfacet theory

Models materials that are composed by many micro facets (smaller than any differential patch) that can be *specular* or *diffuse*.



The *BRDF* of a microfacet surface is given by the following general formula:

$$f(\vec{\omega}_i, \vec{\omega}_o) = \frac{\overbrace{F(\vec{\omega}_i, \vec{\omega}_o)}^{\text{Fresnel coefficient}} \cdot \overbrace{G(\vec{\omega}_i, \vec{\omega}_o)}^{\text{Shadowing}} \cdot \overbrace{D(\vec{\omega}_h)}^{\text{Microfacet Distr.}}}{4|(\vec{\omega}_i \cdot \vec{\mathbf{n}})(\vec{\omega}_o \cdot \vec{\mathbf{n}})|}, \quad \vec{\omega}_h = \frac{\vec{\omega}_i + \vec{\omega}_o}{\|\vec{\omega}_i + \vec{\omega}_o\|} \tag{4.1}$$

$F(\vec{\omega}_i, \vec{\omega}_o)$: the *Fresnel coefficient* determines the fraction of light that is reflected for a given angle between $\vec{\omega}_i$ and $\vec{\omega}_o$.

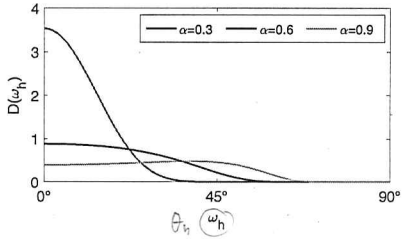
$G(\vec{\omega}_i, \vec{\omega}_o)$: represents the shadowing term, and contains the proportion of microfacets that are not shadowed or masked by the others. As the roughness of the material becomes higher (the micro surface presents more discontinuities), it might happen that for a given incoming direction $\vec{\omega}_i$ some microfacets are shadowed by the ones near them. It may also happen that a microfacet is visible by the light source, but that the path to the camera is occluded by other microfacets. All these factors are contained in the *shadowing* term $G(\vec{\omega}_i, \vec{\omega}_o)$.

$D(\vec{\omega}_h)$: the *Microfacet distribution* determines the fraction of surface that participates in the reflection, i.e. the fraction of microfacets having exactly the right surface normal that reflects the incoming ray in the direction of the outgoing one:

$$\vec{\mathbf{n}} = \frac{\vec{\omega}_i + \vec{\omega}_o}{\|\vec{\omega}_i + \vec{\omega}_o\|}$$

if the normal vector corresponds to the half vector between $\vec{\omega}_i$ and $\vec{\omega}_o$, then the microfacet contributes to the reflection. This quantity is obtained exploiting some principles of statistical physics and computed using the *Beckmann distribution*:

$$D(\vec{\omega}_h) = \frac{1}{\pi\alpha^2 \cos^4\theta_h} e^{-\frac{\tan^2\theta_h}{\alpha^2}}$$



5 Monte Carlo

Rendering consist in solving numerically the *rendering equation*. The integrand is often a nasty and complex function, therefore an analytical solution does not exist. Monte Carlo methods are used to solve this problems:

$$F := \int_\Omega f(\mathbf{x})\,\mathrm{d}\mathbf{x} = \int_\Omega \left[\frac{f(\mathbf{x})}{p(\mathbf{x})}\right] p(\mathbf{x})\,\mathrm{d}\mathbf{x} \approx \frac{1}{N}\sum_{i=1}^{N}\frac{f(\mathbf{x}_i)}{p(\mathbf{x}_i)} = F_N$$

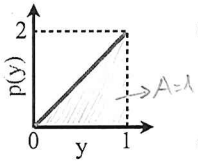
Note that the newly introduced function $p(\mathbf{x})$ allows to importance-sample the integrand: if $f \approx p$, then the resulting integrand is almost a constant function, the samples $\mathbf{x}_i$ are to draw from a probability distribution proportional to p.

Inversion method

The task to draw samples $\mathbf{x}_i$ following an arbitrary distribution p can be solved using the *inversion method*. This method consists in four simple steps:

1) Compute the *cumulative distribution function* (CFD) $P(x) = \int_0^x p(x')\,\mathrm{d}x'$
2) Invert the CFD $P^{-1}(x)$
3) Generate a uniformly random distributed number ξ
4) Evaluate the inverted CFD to obtain a value distributed according to p: $\mathbf{x}_i = P^{-1}(\xi)$

Example: *pdf* we would like to sample: $p(y) = 2y$



Compute the cumulative distribution function $P(y)$ and its inverse $P^{-1}(\xi)$:

$$P(y) = \int_0^y p(y')\,\mathrm{d}y' = y^2, \quad P^{-1}(\xi) = \sqrt{\xi}$$

we can now draw samples that follow the probability density $p(y)$ as follows:

$$\mathbf{y}_i = P^{-1}(\xi), \quad \xi \sim U(0,1)$$

Transformations: given a bijective transformation $T : \mathbf{x}_i \to \mathbf{y}_i$, and given the probability distribution of $\mathbf{x}_i \sim p(x)$, the distribution of

$$\mathbf{y}_i = T(\mathbf{x}_i)$$

can be calculated as follows:

$$p_y(y) = p_y(T(x)) = \frac{p_x(x)}{|det(J_T(x))|}$$

Example: given $p(x) = 1$, and a bijection: $\mathbf{y}_i = T(\mathbf{x}_i) = \sqrt{\mathbf{x}_i}$ calculate the jacobian of the transformation:

$$J_T(x) = \frac{\mathrm{d}y}{\mathrm{d}x} = \frac{1}{2x^{1/2}}$$

then the distribution $p(y)$ is given by:

$$p_y(y) = p_y(\sqrt{x}) = \frac{p_x(x)}{1/(2x^{1/2})} = 2x^{1/2} = 2(y^2)^{1/2} = 2y$$

2D **distribution:** it is possible to draw samples $(\mathbf{x}_i, \mathbf{y}_i)$ from a $2D$ distribution $p(x, y)$.

If $p(x, y)$ (i.e. $p(x, y) = p_x(x)p_y(y)$) then we can independently sample $p_x(x)$ and $p_y(y)$.

Otherwise, we can calculate the marginal distribution $p_x(x)$ as follows:

$$p_x(x) = \int p(x, y)\,\mathrm{d}y$$

and determine the conditional density $p(y|x)$ as:

$$p(y|x) = \frac{p(x, y)}{p(x)}$$

To sample from the *non-separable* $2D$ distribution $p(x, y)$ the procedure is the following:

1) Sample $\mathbf{x}_i \sim p_x(x)$ where $p_x(x) = \int p(x, y)\,\mathrm{d}y$ is the marginal distribution of $p(x, y)$
2) Sample $\mathbf{y}_i \sim p_y(y|x)$ where $p(y|x) = \frac{p(x,y)}{p(x)}$ is the conditional distribution of $p(x, y)$

Example: we want to sample points uniformly inside a unit disc:

$$p(x, y) = \begin{cases} \frac{1}{\pi} & \text{if } x^2 + y^2 < 1 \\ 0 & \text{otherwise} \end{cases}$$

rejection sampling

we sample polar coordinates r, θ where:

$$x = r\cos\theta \quad y = r\sin\theta$$

and we can calculate the *Jacobi matrix* to be:

$$J_T = \begin{bmatrix} \frac{\partial x}{\partial r} & \frac{\partial x}{\partial \theta} \\ \frac{\partial y}{\partial r} & \frac{\partial y}{\partial \theta} \end{bmatrix} \Rightarrow |det(J_T)| = r(\cos^2\theta + \sin^2\theta) = r$$

Therefore we have:

$$p(x, y) = \frac{p(r, \theta)}{r}, \to p(r, \theta) = r \cdot p(x, y) = \frac{r}{\pi}$$

To find the individual density function we calculate the marginal and the conditional distributions:

$$p(r) = \int_0^{2\pi} p(r, \theta)\,\mathrm{d}\theta = 2r, \quad p(\theta|r) = \frac{p(r, \theta)}{p(r)} = \frac{1}{2\pi}$$

Applying the inversion method to the two density functions we can obtain a way to sample points uniformly in a disc:

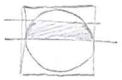
$$\mathbf{r}_i = \sqrt{\xi_1}, \quad \theta_i = 2\pi\xi_2$$

Sampling a sphere: with the above described tools we can now sample points over a sphere:

$$\begin{aligned}\theta &= \cos^{-1}(2\xi_1 - 1)\\ \phi &= 2\pi\xi_2\\ \vec{\omega}_x &= \sin\theta\cos\phi\\ \vec{\omega}_y &= \sin\theta\sin\phi\\ \vec{\omega}_z &= \cos\theta\end{aligned}$$

the algorithm works and is correct, but implies a lot of trigonometric evaluations, this can be avoided exploiting the *Archimedes' Hat-Box theorem*:

> *"..the surface area of a sphere between any two horizontal planes, is equal to the corresponding area of the circumscribing cylinder.."*



following this reasoning we are able to generate samples on a sphere as follows:

$$\begin{aligned}\vec{\omega}_z &= 2\xi_1 - 1\\ r &= \sqrt{1-\vec{\omega}_z^2}\\ \phi &= 2\pi\xi_2\\ \vec{\omega}_x &= r\cos\phi\\ \vec{\omega}_y &= r\sin\phi\end{aligned}$$

where the scalar r is the projection of the point on the cylinder to a point (at the same height) on the sphere.

```
Vector3f Warp::squareToUniformCylinder
    (const Point2f &sample, const Vector2f &h) {
    const float height = std::abs(h(1) - h(0)) * sample.x() + h(0);
    const float phi = 2 * M_PI * sample.y();
    return Vector3f(cos(phi), sin(phi), height);
}
Vector3f Warp::squareToUniformSphere(const Point2f &sample) {
    Vector3f cylinder(squareToUniformCylinder(sample, Vector2f(-1.f, 1.f)));
    const float r = std::sqrt(1.f - cylinder(2) * cylinder(2));
    return Vector3f(r * cylinder(0), r * cylinder(1), cylinder(2));
}
```

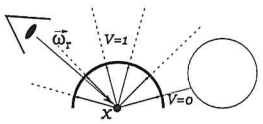
Ambient Occlusion (AO)

Consider a diffuse object illuminated by an ambient overcast sky, then the reflected radiance at a point $\mathbf{x}$ on that surface is given by the reflection equation for diffuse surfaces (3.9):

$$L_r(\vec{x}) = \frac{\rho}{\pi}\int_{H^2} V(\mathbf{x},\vec{\omega}_i)\cos\theta_i\,\mathrm{d}\vec{\omega}_i$$

this integral evaluation can be done via a Monte Carlo approximation of the form:

$$L_r(\vec{x}) \approx \frac{\rho}{\pi N}\sum_{i=1}^{N}\frac{V(\mathbf{x},\vec{\omega}_i)\cos\theta_i}{p(\vec{\omega}_i)}$$



Sampling the direction $\vec{\omega}_i$ uniformly on a hemisphere, we have $p(\vec{\omega}_i) = 1/(2\pi)$, therefore the resulting approximation is of the form:

$$L_r(\vec{x}) \approx \frac{2\rho}{N}\sum_{i=1}^{N} V(\mathbf{x},\vec{\omega}_i)\cos\theta_i \tag{5.1}$$

adopting this uniform sampling technique, the algorithm converges very slowly as there might be a lot of samples that have *zero* contribution (the ones where the visibility term V is zero).

Importance sampling the hemisphere: If we have knowledge about the form of the integrand $f(x)$, we can draw samples according to a distribution $p(x)$ that is *similar* to f:

$$\int f(x)\,\mathrm{d}x, \quad F_N = \frac{1}{N}\sum_{i=1}^{N}\frac{f(\mathbf{x}_i)}{p(\mathbf{x}_i)}$$

with $p(x) = cf(x)$ an the following property we have:

$$1 \overset{!}{=} \int p(x)\,\mathrm{d}x = c\int f(x)\,\mathrm{d}x \to c = \frac{1}{\int f(x)\,\mathrm{d}x}$$

and the integral can be calculated with only one sample:

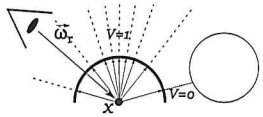
$$\int f(x)\,\mathrm{d}x = \frac{1}{c} = \frac{f(\mathbf{x}_i)}{p(\mathbf{x}_i)}$$

this can be achieved as we assumed we have full knowledge of the integrand function $f(x)$, but, if $pdf p(x)$ is *similar* to the integrand $f(x)$ the variance (therefore the needed number of samples to compute the integral correctly) can be significantly reduced.

Strategy: sample the integral using a *pdf* that is similar to a part of the integrand.

Using a *pdf* proportional to the cosine: using $p(\vec{\omega}_i) = \cos\theta_i/\pi$ we obtain a new estimator for the *reflection-equation*:

$$L_r(\vec{x}) \approx L_r(\vec{x}) \approx \frac{\rho}{\pi N}\sum_{i=1}^{N}\frac{V(\mathbf{x},\vec{\omega}_i)\cos\theta_i}{p(\vec{\omega}_i)} = \frac{\rho}{N}\sum_{i=1}^{N} V(\mathbf{x},\vec{\omega}_i) \tag{5.2}$$



To sample directions $\vec{\omega}_i$ according to a cosine weighted hemisphere one could in principle apply the *inversion method*, but it happens to be, that to sample points in a cosine-weighted fashion, one just have to sample a point on a $2D$ disc, and project it on the hemisphere corresponding to the disc.

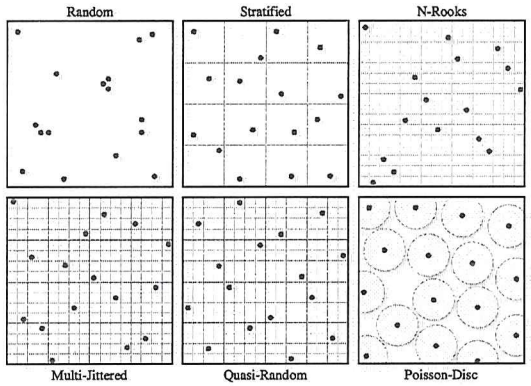
```
Point2f Warp::squareToUniformDisk(const Point2f &sample) {
    const float r = std::sqrt(sample.x());
    const float theta = sample.y() * 2 * M_PI;

    return Point2f(r * cos(theta), r * sin(theta));
}

Vector3f Warp::squareToCosineHemisphere(const Point2f &sample) {
    Point2f disc(squareToUniformDisk(sample));
    return Vector3f(disc.x(), disc.y(),
            std::sqrt(std::max((1e-4f), (1.f - disc.squaredNorm()))));
}
```

Improved random sampling

Instead of drawing random samples (in a $1D$ or $2D$ domain) we can do something better that will help in the reduction of the variance when approximating an integral evaluation using Monte Carlo methods.

- **Random sampling:** trivial sampling, where each sample is independent from the others.
- **Stratification sampling:** divide the sampling domain into disjoint regions, place each sample inside one of the regions. One can prove that this technique does *not* increase the variance, but might reduce it in some cases.

  Stratified sampling requires prior knowledge on the number of samples one has to draw, as the sampling domain has to be split into equal regions.
- **Latin hypercube sampling:** used in $2D$ and higher dimensions. In $2D$ initialize N samples in a diagonal regular manner over the domain (like in a chessboard). As a second step, shuffle the rows of the checkerboard taking the placed samples with the row it belongs to. Last step: shuffle the new columns
- **Multi jittered sampling:** is an improvement of the hypercube sampling, where the sampling domain is at first subdivided into regions (similar to stratification), each of which is treated in a similar way to the latin hypercube sampling.



6 Direct Illumination

The generalized *reflection* equation (that takes into account also the emitted radiance at the evaluation point $\mathbf{x}$) is the following:

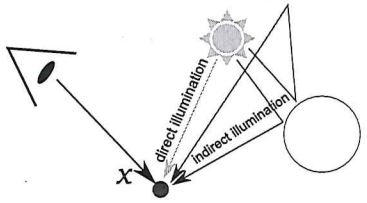
$$L_r(\mathbf{x}, \vec{\omega}_r) = L_e(\mathbf{x}, \vec{\omega}_r) + \int_{H^2} f_r(\mathbf{x}, \vec{\omega}_r, \vec{\omega}_i) L_i(\mathbf{x}, \vec{\omega}_i) \cos\theta_i \, \mathrm{d}\vec{\omega}_i \tag{6.1}$$

An important question that we need to find an answer to, is *how do we calculate* $L_i(\mathbf{x}, \vec{\omega}_i)$*?* This question can have one of the following answers:

Direct illumination: the radiance reflected at point $\mathbf{x}$ is the one that comes directly from an *emitter*:

$$L_i(\mathbf{x}, \vec{\omega}_i) = L_e(\overbrace{r(\mathbf{x}, \vec{\omega}_i)}^{\text{Point which illuminates } \mathbf{x}}, -\vec{\omega}_i))$$

Indirect illumination: the radiance reflected at point $\mathbf{x}$ was scattered around the scene bouncing from surface to surface.



Light path: is the *path* that a light beam follows when scattered in the scene, depending on its origin, the algorithm that generates it is named as follows:

light tracing: the light path starts at the surface of an emitter and is scattered in the scene

path tracing: the light path is generated at the sensor (virtual camera) and is scattered in the scene

bidirectional path tracing: a combination of the above methods, where each generated path is the union between a light path and a camera path

Length of the path: the length of the path (number of segments that composes the path) determines the nature of its contribution to direct/indirect illumination:

length$= 2$**:** direct illumination, direct lighting

length> 2**:** indirect illumination, indirect lighting

Direct Lighting

The reflection equation (6.1) allows us to determine the reflected radiance L_r at a given position $\mathbf{x}$ in a given direction $\vec{\omega}_r$. Note that evaluating the reflected radiance $L_r(\mathbf{x}, \vec{\omega}_r)$ requires an integration over a hemisphere H_2 centered along the normal vector $\vec{\mathbf{n}}$ at $\mathbf{x}$.

Assuming that the surface at location $\mathbf{x}$ is non emissive, the reflected radiance at that point is given by the reflection equation:

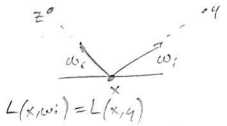
$$L_e(\mathbf{x}, \vec{\omega}_r) = \int_{H^2} f_r(\mathbf{x}, \vec{\omega}_r, \vec{\omega}_i) L_i(\mathbf{x}, \vec{\omega}_i) \cos\theta_i \, \mathrm{d}\vec{\omega}_i$$

The general incident radiance $L_i(\mathbf{x}, \vec{\omega}_i)$ can be replaced by emitted radiance elsewhere in the scene:

$$\boxed{L_i(\mathbf{x}, \vec{\omega}_i) = L_e(r(\mathbf{x}, \vec{\omega}_i), -\vec{\omega}_i)} \tag{6.2}$$

Change of variable: instead of integrating over a hemisphere, therefore over all solid angles $\mathrm{d}\vec{\omega}_i$, we can integrate the integrand over a surface area. The coordinate transformation that needs to be applied in this case is the following:

$$\boxed{\mathrm{d}\vec{\omega} = \frac{\mathrm{d}A \cos\theta_o}{\|\mathbf{y} - \mathbf{x}\|^2}}$$



The reflection equation can be rewritten as:

$$\boxed{L_r(\mathbf{x}, \mathbf{z}) = \int_A f_r(\mathbf{x}, \mathbf{y}, \mathbf{z}) L_e(\mathbf{y}, \mathbf{x}) \overbrace{G(\mathbf{x}, \mathbf{y})}^{\text{Geometry term}} \mathrm{d}A(\mathbf{y})} \tag{6.3}$$

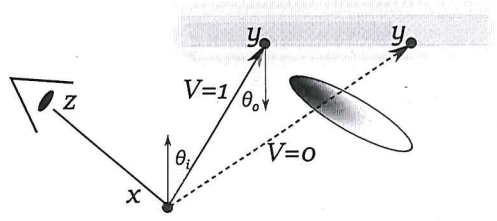
Where the integration is performed over all surfaces of the scene (over every point $\mathbf{y}$), and the *geometry term* is defined as:

$$G(\mathbf{x}, \mathbf{y}) = \overbrace{V(\mathbf{x}, \mathbf{y})}^{\text{Visibility term}} \frac{\cos\theta_i \cos\theta_o}{\|\mathbf{x} - \mathbf{y}\|^2} \qquad V(\mathbf{x}, \mathbf{y}) = \begin{cases} 1 & \text{if visible} \\ 0 & \text{if not visible} \end{cases}$$

Jacobian of the variable transformation

putting everything together we get the following integral equation:

$$L_r(\mathbf{x}, \mathbf{z}) = \int_A f_r(\mathbf{x}, \mathbf{y}, \mathbf{z}) L_e(\mathbf{y}, \mathbf{x}) V(\mathbf{x}, \mathbf{y}) \frac{|\cos\theta_i| |\cos\theta_o|}{\|\mathbf{x} - \mathbf{y}\|^2} \mathrm{d}A(\mathbf{y})$$



where:

$\mathbf{x}$: is the point where the reflection has to be evaluated

$\mathbf{y}$: are the points on the surfaces we are integrating over (usually emitters):

$$\mathbf{y} = r(\mathbf{x}, \vec{\omega}_i)$$

$\mathbf{z}$: position of the camera

the cosine terms in the new reflection equations have the following meaning:

$\cos\theta_i$: is the same as in the old reflection equation (6.1), takes into account that the amount of light that is spread over an infinitesimal patch on the surface depends on the incidence angle θ_i

$\cos\theta_o$: light is emitted following a cosine weighted hemisphere, therefore, if the emission direction is far away from the normal direction at the emission point $\mathbf{y}$, the amount of light going in that direction is reduced.

This cosine term together with the quadratic distance term $\|\mathbf{y} - \mathbf{x}\|^2$ are the *jacobian* of the variable transformation.

Light Sources

- **Point light:** is typically defined by a position $\mathbf{p}$ and an emitted power Φ (or radiant intensity).

  The light emitted by the point light is homogeneous in all directions, and is emitted by exactly one point $\mathbf{p}$: a point light is a *delta* light.

  The evaluation of the reflection equation (6.1) is simplified in this case as we are dealing with a *delta* light:

$$L_r(\mathbf{x},\mathbf{z}) = \int_A f_r(\mathbf{x},\mathbf{y},\mathbf{z}) L_e(\mathbf{y},\mathbf{x}) V(\mathbf{x},\mathbf{y}) \frac{|\cos\theta_i||\cos\theta_o|}{\|\mathbf{x}-\mathbf{y}\|^2} \, dA(\mathbf{y})$$

$$\Downarrow$$

$$L_r(\mathbf{x},\mathbf{z}) = \frac{\Phi}{4\pi} f_r(\mathbf{x},\mathbf{p},\mathbf{z}) V(\mathbf{x},\mathbf{p}) \frac{|\cos\theta_i|}{\|\mathbf{p}-\mathbf{x}\|^2}$$

- **Sphere light:** defined by a position $\mathbf{p}$, a radius r and an emitter power Φ (or emitted radiance). The sphere emitter has a finite surface area of $A = 4\pi r^2$

  When sampling points $\mathbf{y}$ over a sphere light one can follow one of these ideas:

  uniform sample sphere area: sample a position $\mathbf{y}$ over the surface of the sphere. Note that a lot of samples would give a contribution of 0 to the reflection equation as they might lie on the opposite face of the sphere respect to the evaluation point $\mathbf{x}$.

  uniform sample visible cap: sample a position $\mathbf{y}$ on the spherical cap that is visible by the reflection point $\mathbf{x}$. Note that if we uniformly sample the spherical cap, a lot of samples (that lie near to the border of the cap) will emit radiance at an angle θ_o that is big, therefore their contribution to the reflection equation is small.

  → **uniform sample of solid angle:** sample $\mathbf{y}$ according to a uniform distribution of the *solid angle* $\vec{\omega}_i$

  Caution: when evaluating the probability of sampling a point $\mathbf{y}$ on the surface of an emitter, we have to transform it to a probability that is the same as the one used in the reflection equation: angle or area density.

$$p_A(\mathbf{y}) = \frac{\cos\theta_o}{\|\mathbf{y}-\mathbf{x}\|^2} p_\Omega(\vec{\omega}_i) \iff p_\Omega(\vec{\omega}_i) = \frac{\|\mathbf{y}-\mathbf{x}\|^2}{\cos\theta_o} p_A(\mathbf{y})$$

  Example: if we use the angular approach (3.7) to compute the reflected radiance at a given position $\mathbf{x}$, the *pdf* appearing in the denominator of the Monte Carlo approximation (5.2) is the probability density function defined over the solid angles $\vec{\omega}_i$, therefore, given the probability of sampling a point $\mathbf{y}$ on a spherical emitter ($p_A(\mathbf{y}) = \frac{1}{4\pi r^2}$) we have to transform it:

$$p_A(\mathbf{y}) = \frac{1}{4\pi r^2} \rightarrow p_\Omega(\vec{\omega}_i) = \frac{\|\mathbf{y}-\mathbf{x}\|^2}{|-\vec{\omega}_i \cdot \vec{\mathbf{n}}_\mathbf{y}| 4\pi r^2}$$

- **Mesh light:** a mesh can have the additional property of emitting radiance L_e at every point on its surface. The probability of sampling a point $\mathbf{y}$ inside a triangle $\triangle_i$ over the mesh $\mathcal{M} = \cup_i \triangle_i$ is given by the following expression:

$$p_A(\triangle_i) = \frac{A(i)}{\sum_k A(k)}$$

At runtime when sampling a point $\mathbf{y}$ over the emissive mesh, sample a triangle $\triangle_i$ and a point $\mathbf{y}$ in it, compute the *pdf* of choosing that point as follows:

$$p_A(\mathbf{y}) = p_A(\triangle_i) p_A(\mathbf{y}|\triangle_i) = \frac{1}{\sum_k A(\triangle_k)}$$

Multiple strategies

We have seen that the discretization of the reflection equation can be done in various way, choosing different terms of the integrand to importance sample.

Each of the terms that we might be interested to importance sample have pros and cons:

cosine hemisphere sampling: we importance sample the cosine term of the reflection equation choosing directions $\vec{\omega}_i$ according to a cosine weighted hemisphere (instead of a uniform hemisphere). Using this sampling probability, we have:

$$p(\vec{\omega}_i) = \frac{\cos\theta_i}{\pi}$$

which cancels with the cosine term of the integrand

In regions *near* an emitter this technique is very good as a lot of sampled directions $\vec{\omega}_i$ will (luckily) hit the emissive surface .

In regions *far* away from an emitter, we sample mostly directions similar to the surface normal $\vec{\mathbf{n}}$, a lot of samples are similar pointing into the same direction, and a lot of them will miss the emissive surface.

uniform surface area sample points $\mathbf{y}$ on the emissive surface in a uniform way, the incoming radiance L_i is importance sampled, note that all samples have a non zero contribution (if visibility is always $V = 1$)

In regions *near* an emitter this technique performs poorly as the constant area *pdf* is not a good indicator of the effective contributions of each sample: samples that are placed in locations where $\theta_o \approx 90^o$ will be sampled as often a positions $\mathbf{y}$ that have $\theta_o \approx 0^o$, but the first ones will have a very small contribution ($\cos(\theta_o \approx 90^o) \rightarrow 0$) while the ones with $\theta_o \approx 0^o$ have greater contribution ($\cos(\theta_o \approx 0) \rightarrow 1$).

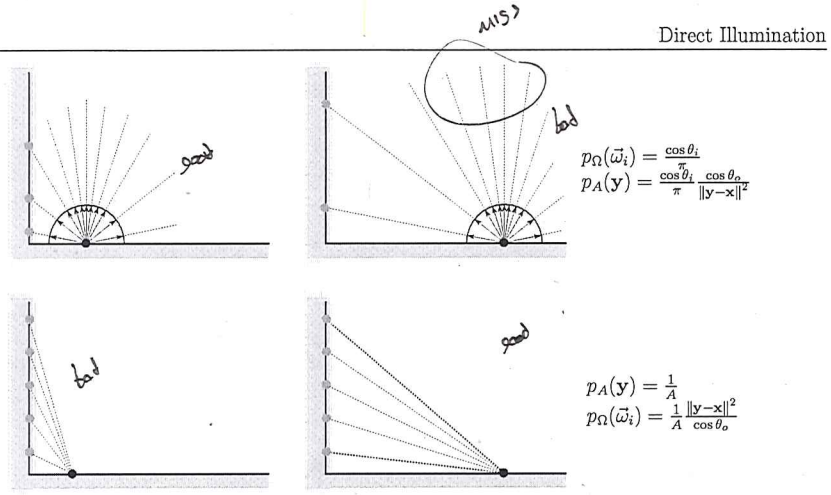
In regions *far* away from the emitter note that all angles θ_o converge together, therefore the uniform positioning of samples $\mathbf{y}$ on the emitter surface is good as all samples have almost the same contribution.

MIS: instead of sampling a single direction/point each time we want to update the Monte Carlo estimator of the reflection equation, we could importance sample both, direction $\vec{\omega}_i$ and a point $\mathbf{y}$, and combine the two results.

This is necessary as *fireflies* often appear in rendered images if the variance is high. This can happen if there are samples that are drawn rarely but have huge contribution:

$$\langle F^N \rangle = \frac{1}{N} \sum_{i=1}^{N} \frac{\overbrace{f(\mathbf{x}_i)}^{\text{Huge contribution}}}{\underbrace{p(\mathbf{x}_i)}_{\text{Small probability}}}$$

A simple averaging of the results obtained using the two sampling strategy does not solve our problem as *variance* is additive.



MIS allows to sample the integrand in such a way that each possible sampled point/direction is well covered by one sampling strategy. Assuming n sampling strategies with corresponding probability function $p_i, i = 1, 2, \ldots, n$, denote with n_i the number of samples taken with strategy i, then $N = \sum_i n_i$, and the *multi-sample* estimator is given by:

$$\langle F^N \rangle = \sum_{i=1}^{n} \frac{1}{n_i} \sum_{j=1}^{n_i} w_i(\mathbf{x}_{i,j}) \frac{f(\mathbf{x}_{i,j})}{p_i(\mathbf{x}_{i,j})} \tag{6.4}$$

where w_i is the weight relative to the i-th sampling strategy. Weight normalization holds: $\sum_{i=1}^{n} w_i(\mathbf{x}) = 1$

There are various methods to choose the weights $w_i(\mathbf{x})$, one of the commonly used ones is the *balance heuristic* one, where each weight w_i is choosen as follows:

$$w_i(\mathbf{x}) = \frac{n_i p_i(\mathbf{x})}{\sum_{i'} n_{i'} p_{i'}(\mathbf{x})}$$

Note that using MIS, the denominator is always big as long as one sampling strategy covers the sample well.

Sampling emitters and BRDF together: to calculate the weights $w_i(\mathbf{x}_i)$ we also need to work using the same *pdf*, usually solid angle *pdf* (sampling and emitter we already get $p_\Omega(\vec{\omega}_i)$ as it is already internally computed).

Small recipe for multiple importance sampling a Monte Carlo estimator $\langle F \rangle \approx \int f(x)\,\mathrm{d}x$ using two *pdf*, $p_1(x)$ and $p_2(x)$ with only one sample per strategy ($n_i = 1$):

$$\begin{aligned} x_1 &= \text{sample}_1(rand) \\ x_2 &= \text{sample}_2(rand) \\ w_1 &= pdf_1(x_1)/(pdf_1(x_1) + pdf_2(x_1)) \\ w_2 &= pdf_2(x_2)/(pdf_1(x_2) + pdf_2(x_2)) \\ \langle F \rangle &= \left(\frac{f(x_1)w_1}{p_1(x_1)} + \frac{f(x_2)w_2}{p_2(x_2)} \right) \end{aligned}$$

Here the implementation of MIS for two sampling strategies $p_A(\mathbf{x})$ and $p_\Omega(\vec{\omega}_i)$:

```
Color3f Li(const Scene *scene, Sampler *sampler, const Ray3f &ray) const override {
 ...
/* Emitter sampling */
/********************/
const Emitter *e = scene->getRandomEmitter(sampler->next1D());

// make a new emitter query record for sampling e from its.p
EmitterQueryRecord eQR(its.p);

// light coming directly from the e to the intersection point its.p
Color3f directEmitter = e->sample(eQR, sampler->next2D());

// evaluate the BSDF for the ray connecting x and y
BSDFQueryRecord bsQR(its.shFrame.toLocal(-ray.d), its.shFrame.toLocal(eQR.wi));
Color3f bsdfColor = its.mesh->getBSDF()->eval(bsQR);

// Perform a visibility check
if(!bsdfColor.isZero() && !scene->rayIntersect(eQR.shadowRay)) {
   float cosTheta_i = its.shFrame.n.dot(eQR.wi);

   // calculate the weights for the multiple importance sampling
   const float pdfEmitterOmegaEmitter = e->pdf(eQR);

   // pdf already calculated in solid angle measure p_Omega(w_i)
   const float pdfMaterialOmegaEmitter = its.mesh->getBSDF()->pdf(bsQR);
   const float wEmitter = pdfEmitterOmegaEmitter /
         (pdfEmitterOmegaEmitter + pdfMaterialOmegaEmitter);
   refl_radiance +=
         wEmitter * directEmitter * bsdfColor * cosTheta * scene->getLights().size();
}
/*     BSDF sampling     */
/*************************/
// get the BSDF at the intersection point
const BSDF *bs = its.mesh -> getBSDF();

BSDFQueryRecord bsQR(its.shFrame.toLocal(-ray.d));

// already weighted with the cosine term cos(theta_i)
Color3f bsdfColor = bs->sample(bsQR, sampler->next2D());

// perform intersection with sampled direction w_i in world space
const Ray3f shadowRay(its.p, its.shFrame.toWorld(bsQR.wo));

if(scene->rayIntersect(shadowRay) && emitIntersection.mesh->isEmitter()) {
   EmitterQueryRecord eQR(its.p, emitIntersection.p);

   // sample the BSDF color
   const Color3f directMaterial = emitIntersection.mesh ->getEmitter()-> eval(eQR);

   // calculate the weights for the multiple importance sampling
   const float pdfEmitterOmegaMaterial =
         emitIntersection.mesh -> getEmitter() -> pdf(eQR);
   const float pdfMaterialOmegaMaterial = bs -> pdf(bsQR);
   const float wMaterial = pdfMaterialOmegaMaterial/
         (pdfMaterialOmegaMaterial+pdfEmitterOmegaMaterial);
   refl_radiance += wMaterial * directMaterial * bsdfColor;
}
return emit_radiance + refl_radiance;
}
```

Environment Lighting

The environment does not have to be modeled explicitly, but can be replaced by one or more images. The images are warped around the $3D$ scene serving as distant light sources.

Map warping: the environmental images can be warped around the $3D$ scene in different ways:

Latitude-Longitude: the map is warped as a sphere around the scene, spherical coordinates θ and ϕ define a location of the map

Cube map: also named *skybox*, the map is warped in the form of a cube around the $3D$ scene.

Other: a lot of other parametrization such as octahedral maps exit

Sample the Environment Map - Marginal and Conditional *pdf*: note that if we perform emitter sampling, we also need to take into account that the environment map is also emissive (often the only emitter in the scene), therefore we need to find a way to importance sample it, in order to sample locations on the map that bring a high contribution to the reflection equation:

$$L_r(\mathbf{x}, \vec{\omega}_r) = L_e(\mathbf{x}, \vec{\omega}_r) + \int_{H^2} f_r(\mathbf{x}, \vec{\omega}_r, \vec{\omega}_i) L_{env}(\vec{\omega}_i) V(\mathbf{x}, \vec{\omega}_i) \cos\theta_i \, \mathrm{d}\vec{\omega}_i \tag{6.5}$$

The task is to sample directions $\vec{\omega}_i$ following a *pdf* that is proportional to the emitted radiance from the environment map:

$$p(\vec{\omega}_i) \propto L_{env}(\vec{\omega}_i) \, \mathrm{d}\vec{\omega}_i$$

Assuming latitude-longitude parametrization of the environment map, we would like to draw samples $\vec{\omega}_i$ from the joint probability distribution:

$$p(\theta, \phi) \propto L_{env}(\theta, \phi) \sin\theta \, \mathrm{d}\theta \, \mathrm{d}\phi$$

note the sine term in θ direction: reduces the amount of light emitted from the environment map at the poles, as the surface area of one pixel near the pole is much smaller than the corresponding surface area of a pixel at the equators.

Integrating a function over a sphere leads to the following:

$$\int_{S^2} f(\vec{\omega}) \, \mathrm{d}\vec{\omega} = \int_0^{2\pi} \int_0^{\pi} f(\theta, \phi) \sin\theta \, \mathrm{d}\theta \, \mathrm{d}\phi \approx \frac{1}{N} \sum_{i=1}^{N} \frac{f(\theta_i, \phi_i) \sin\theta_i}{p(\theta_i, \phi_i)}$$

here we see that if we choose the probability distribution according to $p(\theta, \phi) \propto f(\theta, \phi) \sin\theta$ then the sine will effectively cancel out.

Drawing samples: the following procedure allows to sample directions $\vec{\omega}_i$ according to a probability distribution $p(\theta, \phi) \propto f(\theta, \phi) \sin\theta$:

Step 1: create scalar version $L'(\theta, \phi)$ of $L_{env}(\theta, \phi) \sin\theta$ of the environment map, evaluating the sine term and reducing the RGB color information of each pixel to a scalar value by taking the *maximum coefficient*, the *luminance* or the *average* of the RGB channels.

Step 2: compute marginal probability density function integrating the joint probability along the longitudinal direction:

$$p(\theta) = \int_0^{2\pi} L'(\theta, \phi) \, \mathrm{d}\phi$$

Step 3: compute the conditional probability:

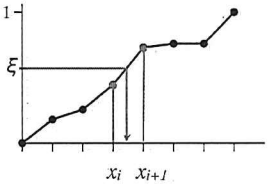
$$p(\phi|\theta) = \frac{p(\theta, \phi)}{p(\theta)}$$

Step 4: draw sample $\theta_i \sim p(\theta)$, then draw sample $\phi_i \sim p(\phi|\theta_i)$

Sample discrete *pdf*: the marginal and conditional probabilities distributions generated in the preprocessing step for sampling an environment map, are discrete.

For a discrete *pdf* it is easy to generalize the concept of *cumulative density function* for the discrete case:

$$CFD(x_i) = \sum_{j=1}^{i} p(x_j)$$

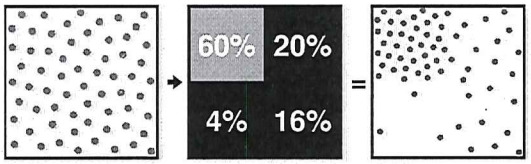


Given a uniform random variable ξ, we can use binary search over the CFD to find x_i such that:

$$CFD(x_i) \leqslant \xi < CFD(x_{i+1})$$

once x_i and x_{i+1} are found, we can determine the exact position x^* having $CFD(x^*) = \xi$ by a linear interpolation.

Sample the Environment Map - Hierarchical sample warping: We can position a set of samples over the environment map following its hierarchical density function. Once the samples have been placed, we can draw one of them randomly as their position is proportional to the density of the environment map.



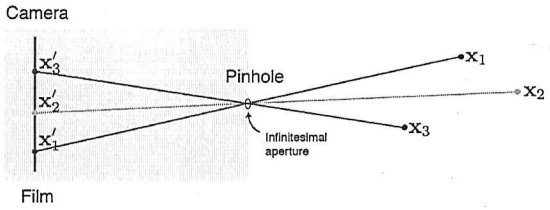
The algorithm that places the samples according to the density of the environment map works recursively on each quadrant of the map.

Advanced Camera Models

- Pinhole camera: the basic camera model used for ray tracing algorithms is the *pinhole* camera lens, which consists in:

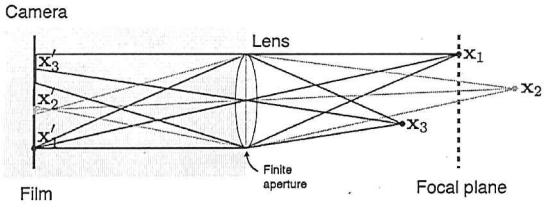
  Camera film: plane where the radiance is cumulated and the image is build

  Pinhole: infinitely small *hole* where the rays pass through and scatter into the scene.



The images resulting with this camera model are innatural as every object in the scene (independently on its distance from the camera) is perfectly sharp.

- **Thin Lens camera:** is an approximation of a real world camera, where the *pinhole* is no longer an infinitesimal aperture, but has a finite aperture.



With the *thin-lens* camera it is possible to introduce effects like the *deph of field*, where objects at different distances from the camera (not on the same perpendicular plane with respect to the camera) might be out of focus.

The implementation of such a camera model has to take into consideration, that the contribution of the color of each pixel is non only given by rays starting at different locations from inside the pixel and passing through the same point in the camera pinhole, but also from rays that (also starting at the same point from inside the pixel) intersect the finite size camera lens at a different position.

7 Global illumination

Light paths

We can express light paths according to the surface interactions that happened to the light when traveling along that path. A path is a set of segments that are linked together by vertices (where the surface interaction happens):

L: a light source

E: the eye/camera

S: specular reflection

D: diffuse reflection

a light path always begins with L and always end with E.

Hackbert's classification: is introduced to facilitate path classification:

k^+: one or more events of type k

k^*: zero of more events of type k

$k?$: zero or one event of type k

$(k|h)$: k or h event

using this syntax we can express path types in a simple way:

Direct illumination: is a light path that starts from an emitter, bounces once over a diffuse or specular surface and then hits the camera sensor:

$$L(S|D)E$$

Indirect illumination: is a light path that starts from an emitter, interacts at least two times with a surface, and terminates in the camera sensor:

$$L(S|D)(S|D)^+E$$

An algorithm that can handle full global illumination has to handle paths of type:

$$L(S|D)^*E$$

and to render caustics it also has to handle paths of the form:

$$LS^+DE$$

Rendering equation

As already seen before, the rendering equation that we are trying to solve is given by:

$$\underbrace{L_o(\mathbf{x},\vec{\omega}_o)}_{\text{outgoing radiance}} = \underbrace{L_e(\mathbf{x},\vec{\omega}_o)}_{\text{emitted radiance}} + \underbrace{L_r(\mathbf{x},\vec{\omega}_o)}_{\text{reflected radiance}}$$

that can explicitely be rewritten as:

$$\boxed{L_o(\mathbf{x},\vec{\omega}_o) = L_e(\mathbf{x},\vec{\omega}_o) + \int_{H^2} f_r(\mathbf{x},\vec{\omega}_i,\vec{\omega}_o) L_i(\mathbf{x},\vec{\omega}_i)\cos\theta_i \,\mathrm{d}\vec{\omega}_i} \tag{7.1}$$

Assuming non participating media we can state that radiance remains constant along a segment of its light path , we can therefore relate the incoming radiance $L_i(\mathbf{x}, \vec{\omega}_i)$ to the outgoing radiance L_o at an other position $\mathbf{x}' = r(\mathbf{x}, \vec{\omega}_i)$:

$$L_i(\mathbf{x}, \vec{\omega}_i) = L_o(r(\mathbf{x}, \vec{\omega}_i), -\vec{\omega}_i) \tag{7.2}$$

The rendering equation can be rewritten substituting the incoming radiance with the corresponding outgoing radiance (drop 'o' notation for the *outgoing* radiance and direction):

$$L(\mathbf{x}, \vec{\omega}) = L_e(\mathbf{x}, \vec{\omega}) + \int_{H^2} f_r(\mathbf{x}, \vec{\omega}_i, \vec{\omega}) L(r(\mathbf{x}, \vec{\omega}_i), -\vec{\omega}_i) \cos\theta_i \, \mathrm{d}\vec{\omega}_i$$

The surface area form of the above recursive equation is given by:

$$L(\mathbf{x}, \mathbf{z}) = L_e(\mathbf{x}, \mathbf{z}) + \int_A f_r(\mathbf{x}, \mathbf{y}, \mathbf{z}) L(\mathbf{y}, \mathbf{x}) G(\mathbf{y}, \mathbf{x}) \, \mathrm{d}A(\mathbf{y})$$

where the notation is the same used in the previous sections.

Solution to the rendering equation: closed forms for the rendering equations are very hard to find as we are dealing with a high dimensional integral equation, whose integrand might be discontinuous (visibility term). Gaussian quadrature is inconvenient as we are working in high dimensions.

Recursive Monte Carlo Ray Tracing

In a ray tracing algorithm, often direct and indirect illumination are evaluated separately: direct illumination often performs better with an emitter sampling technique, indirect illumination is calculated gathering new rays from the primary intersection point:

$$\begin{aligned} L(\mathbf{x}, \vec{\omega}) &= L_e(\mathbf{x}, \vec{\omega}) + L_d(\mathbf{x}, \vec{\omega}) + L_i(\mathbf{x}, \vec{\omega}) \\ &= L_e(\mathbf{x}, \vec{\omega}) + \underbrace{\int_A \dots L_e(\mathbf{y}, \mathbf{x}) \dots \mathrm{d}A(\mathbf{y})}_{\text{direct lighting}} + \underbrace{\int_{H^2} \dots L(r(\mathbf{x}, \vec{\omega}_i), -\vec{\omega}_i) \dots \mathrm{d}\vec{\omega}_i}_{\text{indirect lighting, recursive formulation}} \end{aligned} \tag{7.3}$$

note that this formulation is recursive: in the evaluation of indirect illumination we make use of $L(r(\mathbf{x}, \vec{\omega}_i), -\vec{\omega}_i)$. A pseudo code for evaluating equation (7.3) presented above is the following:

Algorithm 1 Monte Carlo global illumination evaluation

```
1: function L(x, ω)
2:     for all lights e do                          ▷ Direct illumination via emitter sampling
3:         L_d += contribution from light e
4:     end for
5:     for all N indirect sample rays do            ▷ indirect illumination
6:         ω_i = BRDF.sample(rand)
7:         L_i += BRDF.color() · dot(n, ω_i)/(N · p_Ω(ω_i)) · L(r(x, ω_i), −ω_i)
8:     end for
9:     if last bounce specular then                 ▷ Prevent double counting
10:        return L_e + L_d + L_i
11:    else
12:        return L_d + L_i
13:    end if
14: end function
```


note that at line 9 of algorithm 1 we need to differentiate between two cases to decide what to return. This is due to the fact that if the last bounce was specular, then it was impossible (because of the delta-like BSDF) that the current surface (if emissive) gave a contribution. On the other hand, if the surface was diffuse, then the current emissive surface (that was reached by a shadow ray in the previous bounce) already gave its contribution.

Remark: note that the above presented algorithm is has a recursive nature, and its complexity is exponential: for each iteration in the recursion we call $L(\mathbf{x}, \vec{\omega})$ N times (see 1.5). To avoid the explosion in number of shadow rays we can reduce the algorithm to what is called the *path tracing algorithm*.

Path Tracing

Instead of sampling N shadow rays at each shadowing point $\mathbf{x}$ we can reduce the complexity of the algorithm by simply sample one new direction $\vec{\omega}_i$.

Algorithm 2 Path Tracing global illumination evaluation

```
1: function L(x, ω)
2:     for all lights e do                          ▷ Direct illumination via emitter sampling
3:         L_d += contribution from light e
4:     end for
5:     ▷ only one shadow ray to estimate indirect illumination
6:     ω_i = BRDF.sample(rand)
7:     L_i += BRDF.color() · dot(n, ω_i)/p_Ω(ω_i) · L(r(x, ω_i), −ω_i)
8:     if last bounce specular then                 ▷ Prevent double counting
9:         return L_e + L_d + L_i
10:    else
11:        return L_d + L_i
12:    end if
13: end function
```

note that in this version of the algorithm, we only sample one new direction $\vec{\omega}_i$ per recursion step (see 2.5), the algorithm does not explode in the number of paths that we need to follow but is still infinite: it does not terminate.

Russian Roulette: how do we stop the recursion of the above presented algorithm? Truncating the recursion at a fixed number of recursion steps is not a good idea as it introduces *bias*.

Russian roulette is a probabilistic termination rule:

1) choose a termination probability $q \in [0, 1]$

2) in each recursion iteration do the following:

- **2a)** generate a random number ξ
- **2b)** if $\xi > q$ then continue the recursion following the new path $L(r(\mathbf{x}, \vec{\omega}_i), \vec{\omega}_i)$
- **2c)** else if $\xi \leq q$ terminate the recursion returning from the function. Note that to be consistent, in case of recursion truncation we need to rescale the indirect illumination as follows:

$$L_i \to \frac{L_i}{1 - q}$$

The improved path tracing algorithm is the following:

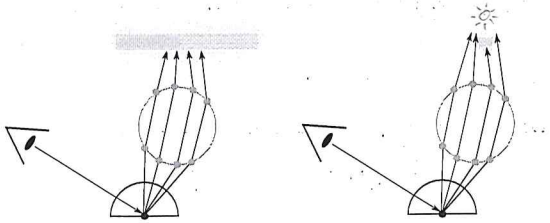
Algorithm 3 Path Tracing global illumination evaluation with RR termination

```
1: function L(x, ω⃗)
2:     for all lights e do                    ▷ Direct illumination via emitter sampling
3:         L_d += contribution from light e
4:     end for
5:     if rand > q then
6:         ω⃗_i = BRDF.sample(rand)
7:         L_i += BRDF.color() · dot(n⃗, ω⃗_i) / p_Ω(ω⃗_i) · L(r(x, ω⃗_i), −ω⃗_i)
8:     end if
9:     if last bounce specular then
10:        return L_e + L_d + L_i/(1 − q)     ▷ Normalize indirect illumination
11:    else
12:        return L_d + L_i/(1 − q)           ▷ Normalize indirect illumination
13:    end if
14: end function
```

Remark: note that to sample direct illumination one can also use the MIS technique in order to reduce the variance.

Path tracing caustics: Path tracing is a pretty straight forward algorithm that is able to render scene that need ray scattering to be reproduced correctly (dielectrics, scattering,...).

Nevertheless path tracing also fails miserably in some situations, where delta lights (or small emitters) have a huge contribution in the formation of visual effects, like caustics, where reflection and refraction generated by delta-like **BRDF**s plays an important role.



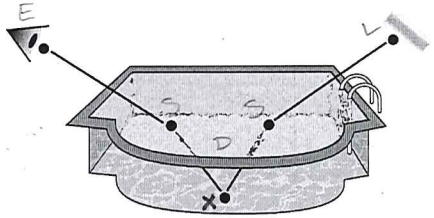
If the emitter is small (figure on the right), then it is almost impossible to importance sample a direction $\vec{\omega}_i$ at the shading point in order to hit the emitter: a lot of sampled directions $\vec{\omega}_i$ give a zero contribution to the caustic that should appear below the dielectric sphere.

If the emitter is a spot-light, then path tracer based algorithms will always give a black image as the probability of hitting a delta emitter sampling a random direction is $p = 0$.

Bidirectional Path Tracing

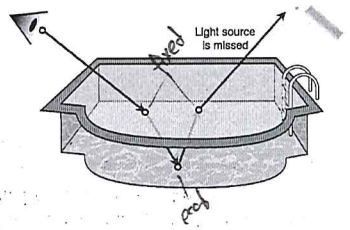
is a combination of path and light tracing, where light paths start at both, emitter and camera and are linked together in the middle of the scene.

This combination allows to render images with much less noise than using path or light tracing separate. Still, this technique is not yet robust enough for paths of the type $LSDSE$:

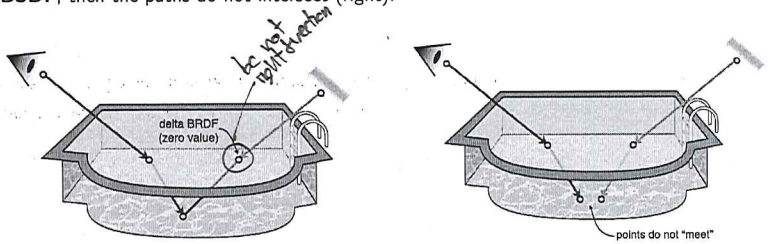


The problem lies in the fact that for a path to exist from the *eye* to the *emitter*, it has to pass once through a delta **BSDF**, bounce on a diffuse surface and pass once again through the delta **BSDF**: the probability of a camera path to hit exactly the point $\mathbf{x}$ on the diffuse surface that has been hit by the emitter path is $p \ll 1$.

Path tracing fails: as the path has to follow the **BSDF** of the specular surface, it has low probability of hitting the emitter.



Bidirectional tracing fails: as the linking between the camera and the emitter path makes the delta **BSDF** of the specular surface to zero (left). If the linking between the paths is done after *passing through* the specular **BSDF**, then the paths do not intersect (right).



8 Many Lights Rendering

The rendering equation (7.1) is composed by three parts:

emission: L_e consists in the amount of radiance emitted at the shading point $\mathbf{x}$ in direction $\vec{\omega}_o$

direct illumination: L_d is the amount of radiance reflected at point $\mathbf{x}$ due to direct illumination (path $L(D|S)E$), easy to evaluate performing emitter sampling (or MIS)

indirect illumination: L_i corresponds to the amount of reflected light that reflects at the shading point $\mathbf{x}$ due do indirect illumination, i.e. to a light path that has already bounced once in the scene. Can be hard to compute and is usually not cheap as direct illumination

Many Light Rendering is a technique that allows to estimate indirect illumination in a fast and cheap way, without incurring in too much variance.

Virtual Point Light

Indirect illumination is calculated considering paths of the type $L(S|D)(S|D)^+E$, where light scatters at least two times in the scene.

A fast approximation of the global illumination term $L_i(\mathbf{x}, \vec{\omega}_i)$ can be found by the following intuition:

- **Preprocess:** distribute virtual point light (VPL) around the scene at the vertices of a given number of light paths. The power of each VPL is given by the power of the original emitter attenuated by the cumulated **BSDF** of the surface interaction on the path.
- **Radiance evaluation:**

  direct lighting: evaluate as usual sampling the emitters (MIS also possible)

  indirect lighting: approximated by sampling the VPL as if they where emitters

Note that in the preprocessing step one could introduce a criteria to select VPL that will have a huge contribution, while deleting the ones that will have almost no influence in the rendering of the scene.

VPL can also be generated making use of the *bidirectional path tracing* algorithm, where VLP are positioned at the second vertex starting from the camera.

Lighting with VPL: images rendered using this technique as it is, will present a lot of *splotches* in regions near the VPL: the *geometric term* of the reflection equation (6.3) has a singularity at $\mathbf{x} = \mathbf{y}$:

$$G(\mathbf{x}, \mathbf{y}) = V(\mathbf{x}, \mathbf{y}) \frac{\cos\theta_o \cos\theta_i}{\|\mathbf{y} - \mathbf{x}\|^2}$$

Note that for a shading point $\mathbf{x}$ lying on the same plane (wall) of a VPL this is not a problem as the cosine terms in the numerator of the *geometric term* are zero. Problems are present in regions where the scene has corners: the shading point $\mathbf{x}$ can be really near the VPL location $\mathbf{y}$, and the cosine terms are not zero.

To solve this problem we can introduce a cutoff of the geometry term:

$$\boxed{G_b(\mathbf{x}, \mathbf{y}) = \min\{G(\mathbf{x}, \mathbf{y}), b\}}$$

this trick is extremely simple and computationally cheap, introduces *bias* and makes cavities appear darker than they should (as it removes short light transport).

As every shading location samples the same VPL to evaluate indirect illumination, a visual error shows up under the form of structured artifacts: *indirect* shadows generated by each VPL are present in the scene if too few VPL are used.

Note that in theory, using an infinite number of VPL (as the light path is infinite and scatters continuously around the scene), *many lights* based algorithms are unbiased.

Virtual Spherical Lightemitters:VSL

instead of concentrating the radiance of a path into a point light, we can bound it inside an area emitter like a sphere light.

The differences between VSL and VPL are the following:

singularities: in VSL the energy is blurred over a surface, not concentrated in a single point, no need for clamping

blurring: introduces *bias*

complexity: evaluating the incoming radiance from a point light is much easier that fro a spherical light, as for VSL we have to integrate the incoming radiance over its surface.

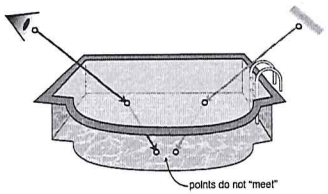
Applications

Many light rendering is a technique used in real time application visualization as it allows fast evaluation of indirect illumination.

It has the disadvantage that for subsequent frames (in a movie) point lights change their position in space, resulting in a flickering effect of the video as different surfaces are illuminated differently in different frames.

9 Photon Mapping

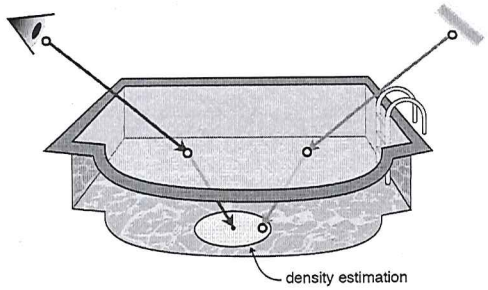
The need of a new method to compute indirect illumination correctly is high, because as we have seen so far, *unbiased* methods fail in computing indirect illumination of path of the type $LSDSE$.



Photon Mapping is a *biased* method that allows to estimate efficiently indirect illumination also for this kind of path.

Starting from the observation that bidirectional path tracing fails in joining camera and emitter path in $LSDSE$ situations.

We can relax the constraint of linking paths exactly allowing a maximal distance between their heads, that, if respected counts as a valid linking. Reinterpreting the relaxation above, we can evaluate the indirect lighting at a shading point $\mathbf{x}$ by integrating the incoming radiance L_i over all points $\mathbf{y}$ that are in the neighborhood of $\mathbf{x}$:



Backward Ray Tracing

Photons are emitted from lights and are scattered around the scene, each time they hit a surface their power is reduced due to the participation of the *BRDF*.

Photons can be stored in a *illumination map*, which in the end of the emission preprocessing step, contains information about the *irradiance* at any point $\mathbf{y}$ of the scene (*irradiance:* number of photons hitting a differential patch of a wall per second, divided by the size of the patch).

When evaluating the reflected radiance at a point $\mathbf{x}$ sample the emitters for direct lighting L_d, lookup indirect lighting in *illumination map*.

Note that *illumination maps* require to parametrize each surface of the scene in order to store irradiance informations, the number of photons stored in the *illumination map* determines the type of noise that will appear in the resulting image: too many photons (high resolution) will generate high-frequency noise, too few photons will generate a blurred image.

Photon Mapping Algorithm

The *Photon Mapping Algorithm* is a two-pass algorithm:

Step 1: Trace photons from all light sources into the scene, store them in a *photon map*

Step 2: Trace rays from the camera, at shading location perform direct lighting computation by emitter sampling, indirect illumination by using the photons stored in the *photon map*

Photon emission: each photon is emitted by a light source at a given position $\mathbf{x}_p$, in a given direction $\vec{\omega}_p$ and with a given power Φ_p.

The position $\mathbf{x}_p$ on the emitter surface is chosen sampling a probability density function $p(\mathbf{x}_p)$, the direction $\vec{\omega}_p$ is chosen from a conditional density function $p(\vec{\omega}_p|\mathbf{x}_p)$. The photon power Φ_p is determined as follows:

$$\Phi_p = \frac{1}{M}\frac{L_e(\mathbf{x}_p, \vec{\omega}_p)}{p(\mathbf{x}_p)p(\vec{\omega}_p|\mathbf{x}_p)}, \quad M = \text{number of emitted photons}$$

note that if the *pdf*s of position and direction are proportional to the emission at a given point $\mathbf{x}_p$ and respective direction $\vec{\omega}_p$:

$$p(\mathbf{x}_p) = \frac{\int_{H^2} L_e(\mathbf{x}_p, \vec{\omega})\,\mathrm{d}\vec{\omega}}{\int_A \int_{H^2} L_e(\mathbf{x}, \vec{\omega})\,\mathrm{d}\vec{\omega}\,\mathrm{d}\mathbf{x}}, \quad p(\vec{\omega}_p|\mathbf{x}_p) = \frac{L_e(\mathbf{x}_p; \vec{\omega}_p)}{\int_{H^2} L_e(\mathbf{x}_p, \vec{\omega})\,\mathrm{d}\vec{\omega}}$$

then the photon power Φ_p can be simply computed as follows:

$$\Phi_p = \frac{\Phi}{M}$$

Note that with a constant cosine-weighted emission L_e over the emitter surface, the probabilities simplify to:

$$p(\mathbf{x}_p) = \frac{1}{A}, \quad p(\vec{\omega}_p|\mathbf{x}_p) = \frac{\cos\theta_p}{\pi}$$

Photon emission examples:

Point Light: assuming isotropy (light is emitted in all directions with the same intensity), all photons have position $\mathbf{x}_p = \mathbf{p}$, and their direction is sampled from a uniform spherical distribution $p(\vec{\omega}_p) = \frac{1}{4\pi}$

Spot Light: the position $\mathbf{x}_p$ is the emitter position $\mathbf{p}$, a direction $\vec{\omega}_p$ is sampled uniformly inside the spherical cap that generates the light cone: $p(\vec{\omega}_p) = \frac{1}{2\pi(1-\cos\theta_a)}$, where θ_a is the aperture of the light cone

Diffuse Square Light: generate the position over the surface area by sampling a random point $\mathbf{x}_p$ uniformly on the emitter, sample a direction $\vec{\omega}_p$ with a cosine-weighted direction over the hemisphere defined by the surface normal $\vec{\mathbf{n}}$

Algorithm 4 Photon Emission

```
1: function GeneratePhotonMap
2:     while not enough photons do
3:         (l, p_l) = chooseRandomEmitter()   ▷ p_l: probability of choosing emitter l
4:         (x_p, ω_p, Φ_p/p_l) = emitPhotonFromLight(l)
5:         tracePhoton(x_p, ω_p, Φ_p)
6:     end while
7: end function
```

Note that the photon power Φ_p is divided by p_l, this corresponds to multiplying its power by the number of emitters in the scene.

Once the photon is generated, it is scattered in the scene following its initial direction $\vec{\omega}_p$, once the photon hits a **diffuse** surface, part of it is absorbed and stored in the photon map. Note that no photon is stored over a *specular* surface, as the color of a specular surface is resolved with path tracing.

Every time a photon bounces on a surface, its power Φ_p is reduced according to the **BSDF** of the intersection point and the new direction $\vec{\omega}'_p$ chosen sampling the **BSDF**:

$$(\vec{\omega}'_p, pdf) = \text{SAMPLEBSDF}(-\vec{\omega}_p), \quad \Phi'_p = \Phi_p \cdot \cos\theta'_i \cdot \text{BSDF}(-\vec{\omega}_p, \vec{\omega}'_p)/pdf$$

if the photon survives a russian roulette stopping criteria, then the photon is scattered in the scene once again with the new parameters. The russian roulette stopping criteria allows to terminate the photon scattering inside the scene with termination probability:

$$q_p = 1 - \min\left\{1, \frac{\Phi'}{\Phi}\right\}$$

also here, if a photon survives the russian roulette (drawing a random number $\xi \geq q_p$), its power needs to be rescaled in order to have normalization:

$$\Phi' = \frac{\Phi'}{1 - q_p}$$

note that if a photon survives the russian roulette, its new power is given by:

$$\Phi' = \frac{\Phi'}{1 - (1 - \min\left\{1, \frac{\Phi'}{\Phi}\right\})} = \Phi$$

the scattered photon has the same power as the incoming photon.

Example: if 300 photons with power $\Phi_p = 1W$ hit a diffuse wall with reflectance probability $p = 0.5$ we will have 150 photons with power $\Phi'_p = 1W$ instead of 300 photons with half power $\Phi'_p = 0.5W$.

A photon is stored in the photon map as follows:

```
struct Photon
{
    float position[3];
    float power[3];
    float direction[3];
};
```

Indirect light evaluation: the second stage of the algorithm consists in a normal ray tracing (ray starting at the camera position), where at every intersection point $\mathbf{y}$ in the scene, we can evaluate indirect illumination querying the photon map:

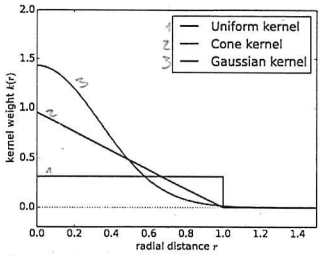
Algorithm 5 Photon Map Query

1: **function** LI($\mathbf{x}, \vec{\omega}$)
2: ▷ Find intersection point $\mathbf{y}$ in the scene
3: ▷ Evaluate direct illumination via emitter sampling
4: $Photons[\,] = k$ nearest photons to $\mathbf{y}$ in the photon map
5: $L_r(\mathbf{y}, \vec{\omega}) = \sum_{p=1}^{k} f_r(\mathbf{y}, \vec{\omega}_p, \vec{\omega})\Phi_p/A$ ▷ A: surface of density evaluation region
6: **end function**

Assuming a density evaluation region A of a circle with radius r, we have the indirect light estimation:

$$L_i(\mathbf{y}, \vec{\omega}) = \sum_{p=1}^{k} f_r(\mathbf{y}, \vec{\omega}_p, \vec{\omega})\frac{\Phi_p}{\pi r^2} \tag{9.1}$$

It is indeed possible to define other *kernels* that weight the contribution of each photon according to its distance from the shading point $\mathbf{y}$: a photon p_{far} that is further away from $\mathbf{y}$ than a near one p_{near} will contribute less to the indirect illumination at $\mathbf{y}$.



Each kernel that is used in the evaluation of indirect illumination has to satisfy normalization:

$$\int_0^{2\pi}\int_0^{r} w(r')r'\,\mathrm{d}r'\,\mathrm{d}\phi = 1 \tag{9.2}$$

- **Uniform kernel:** every photon found in the local neighborhood of the shading point $\mathbf{y}$ is weighted equally. To have normalization (when using a different neighborhood radius r) the weight given to each photon is the following:

$$w_p = \frac{1}{\pi r^2}$$

the computation of the indirect illumination L_i is easy:

$$L_i(\mathbf{y}, \vec{\omega}) = \sum_{p=1}^{k} f_r(\mathbf{y}, \vec{\omega}_p, \vec{\omega})\frac{\Phi_p}{\pi r^2}$$

Cone kernel: the weight associated with each photon inside the local neighborhood defined by a radius r decays linearly with the distance $\|\mathbf{x}_p - \mathbf{y}\|$ from the shading point $\mathbf{y}$:

$$w_p(r) = h \cdot \left(1 - \frac{\|\mathbf{x}_p - \mathbf{y}\|}{r}\right), \quad h = \frac{3}{\pi r^2}$$

note that when the distance from the photon to the shading point $\mathbf{y}$ is equal the radius r of the neighborhood, then the weight collapses to $w_p(r) = 0$, a photon positioned exactly at the shading point contributes with a weight of $w(r) = h$, where h is the *height* of the cone, which is chosen such that the kernel is normalized (see (9.2))

$$L_i(\mathbf{y}, \vec{\omega}) = \sum_{p=1}^{k} f_r(\mathbf{y}, \vec{\omega}_p, \vec{\omega})\frac{3}{\pi r^2}\left(1 - \frac{\|\mathbf{x}_p - \mathbf{y}\|}{r}\right)$$

Gaussian kernal: Also possible to implement, weight $w_p(r)$ is given following a gaussian weighting distribution of the form:

$$w_p(r) \propto e^{-\frac{1}{2}\frac{\|\mathbf{x}_p - \mathbf{y}\|^2}{\sigma^2}}$$

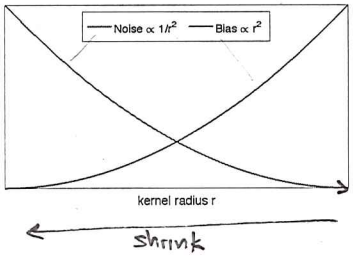
where σ has to be chosen such that the gaussian distribution decays fast enough to zero for distances $\|\mathbf{x}_p - \mathbf{y}\|$ that approach the neighborhood radius r

```
Color3f Li(const Scene *scene, Sampler *sampler, const Ray3f &ray) const override {
    ...
    // scatter ray until we are on a diffuse surface
    // perform emitter sampling as direct illumination
    Ld = ...
    // perform photon-mapping as indirect illumination
    Li += itsColor * m_kernel->getRadiance(y, -ray.d, m_photonMap)/m_emittedPhotonsPerLight;
    return (Ld + Li)*bsColor;
}

// uniform kernel
virtual Color3f getRadiance(const Point3f& y, const Vector3f& wi,
      std::unique_ptr<PhotonMap> const & photonMap) override {
    Color3f radiance(0.f);
    // find the photons lying in a neighborhood r from y
    std::vector<uint32_t > results; photonMap -> search(y, r, results);

    for(unsigned int photon = 0; photon < results.size(); ++photon)   {
        const Photon &P = (*photonMap)[results[photon]];
        radiance += P.getPower();
    }
    return radiance / (πr²);
}

// cone kernel
virtual Color3f getRadiance(const Point3f& y, const Vector3f& wi,
std::unique_ptr<PhotonMap> const & photonMap) override {
    Color3f radiance(0.f);
    // find the photons lying in a neighborhood r from y
    std::vector<uint32_t > results; photonMap -> search(y, r, results);

    for(unsigned int photon = 0; photon < results.size(); ++photon)   {
        const Photon &P = (*photonMap)[results[photon]];

        float d = (y - P.position).norm();
        float h_d = h * (1.f - d/r);
        radiance += h*P.getPower();
    }
    return radiance;
}

// gaussian kernel
virtual Color3f getRadiance(const Point3f& y, const Vector3f& wi,
std::unique_ptr<PhotonMap> const & photonMap) override {
    Color3f radiance(0.f);
    // find the photons lying in a neighborhood r from y
    std::vector<uint32_t > results; photonMap -> search(y, r, results);

    for(unsigned int photon = 0; photon < results.size(); ++photon) {
        const Photon &P = (*photonMap)[results[photon]];

        const float dist2 = (y - P.position).squaredNorm();
        radiance +=  std::exp(-0.5f*(dist2/σ²))* P.getPower();
    }
    // normalize the weights such that (9.2) is almost satisfied
    return radiance * m_normalization;
}
```


Progressive Photon Mapping

One of the drawbacks of *photon mapping* is that during the preprocessing step (that might take a lot of time) a lot of memory is used (cache information about photon position) and no preview of the rendered image is shown.

Progressive photon mapping is an extension of the classical *photon mapping* algorithm. The key idea of this extension is to progressively *shrink* the kernel estimation radius r over which the neighborhood query is executed.



As one can imagine, the larger the kernel radius r is, the bigger the *bias* is (as more blurring appears). A too small radius is also non convenient because (beside to increasing the number of required photons in the photon map) it generates high-frequency noise.

Algorithm 6 Progressive Photon Mapping

$I = null; \alpha \in [0, 1]$ ▷ Shrinking parameter
GeneratePhotonMap()
for Iteration $i = 0$ to $i = maxIt$ **do**
    $I_i = null$
    ▷ Perform Photon Mapping with a given radius r_i
    $I_i = \text{LI}(\ldots, r_i)$
    $r_{i+1}^2 = \frac{i+\alpha}{i+1} r_i^2$ ▷ Update kernel radius
    $I \mathrel{+}= I_i$
end for
$I \mathrel{/}= maxIt$

Convergence Analysis: Using the progressive photon map technique, each time we shrink the kernel radius r we increase *noise* in the image I_i, but reduce its *bias*. Averaging the total cumulated *noise* and *bias* in the image I they both converge towards zero.

A big value of the shrinking parameter α allows a slow shrinking of the radius, while a small value pushes the radius towards zero really fast. A value of $\alpha = 2/3$ minimizes the *mean squared error MSE*.

Photon Relaxation

Photon maps suffer from local density peaks and local empty spaces due to:

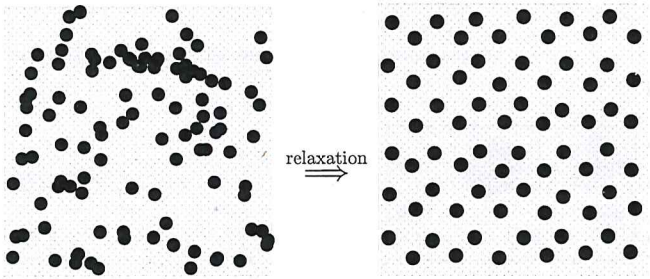
1) stochastic processes (e.g. **BRDF** sampling)

2) tracing decoherence

A big variance in the photon density in the photon map causes noise and needs to be queried with a bigger kernel radius r to avoid regions where no photon is found in the neighborhood of a shading point.

We can avoid local empty spaces in the photon map by introducing a repulsion force between them. Applying the repulsion force on each photon (force generated by its k nearest neighbors) each photon is displaced.

After this process, the photon configuration is relaxed and is more even.



10 Acceleration Data Structures

Ray-surface intersection queries are the core of any ray tracing algorithm. As such, this kind of operation need to be done in a very efficient and fast way.

A *brute-force* method to perform ray-surface intersection for an image of size $N_x \times N_y$ with a scene containing N_o primitives (triangles) would have a rendering complexity of:

$$\mathcal{O}(N_x N_y N_o)$$

assuming a single ray per pixel we have a linear complexity $\mathcal{O}(N_o)$ in the number of primitives.

A way to overcome this linear scalability, we need to introduce *accelerating data structures* that allow to perform ray-surface intersection tests much faster, i.e. without the need to check each primitive in every ray-surface intersection query.

Introducing accelerating data structures we can reject groups of primitives that we are sure will not be intersected by the ray. To be capable of rejecting a large fraction of primitives from the ray-surface intersection query, we can exploit one of the following two ideas:

spatial subdivision: the virtual space of the scene is subdivided in (regular or adaptive) disjoint subregions, each having a list of the primitives it contains. A ray-primitive intersection test is performed only with the primitives contained in the subregions where the ray passes through

hierarchical bounding volumes: a hierarchical structure is build upon the objects, each object (set of primitives) is therefore partitioned into a hierarchy of disjoint sets, each of which belongs to the same hierarchical tree. A ray-primitive intersection test only happens if the ray intersected all *ancestors* of the primitive in the hierarchy.

Spatial Subdivision

To accelerate the process of finding the intersection of a ray $\mathbf{r}(t) = \mathbf{o} + \mathbf{d}t$ with a primitive in the scene one can subdivide the scene into subregions, each containing a small part of the whole scene. Each subregion (in the case of regular parallelepipeds, voxels) contains a reference to the primitives that lie inside of it.

When an intersection query is executed, the spatial grid is queried for ray-subspace intersections (ray-voxel in the case of a regular grid), and only the primitives lying inside the intersected regions are further investigated for a possible ray-surface intersection.

All primitives belonging to spatial regions that were not intersected by the ray are not taken in consideration for the ray-surface intersection test.

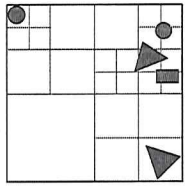
Note that the ray-subregion intersection test can be performed in a meaningful order, i.e. considering first the subregions near the origin of the ray, and later the ones further away, in this way, if a real ray-surface intersection happens, all candidate primitives belonging to regions behind the intersection point are not queried.

Remark: regular spatial subdivision accelerating data structures can suffer from poor performance in the case where the primitives are not equally distributed in the virtual $3D$ space. In the case where a lot of subregions are empty, and one of them contains a lot of primitives (a large scene containing a small detailed mesh), then the overhead of querying the accelerating data structure is high, as a lot of subregions do not

contain any primitive; when the dense filled subregion is hit, then a lot of primitives have to be checked for the ray-surface intersection. Adaptive spatial subdivision accelerating data structure do not suffer from this type of problems.

- Octrees: Also this is a hierarchical spatial subdivision data structure. Similarly to the *KD-tree* to build the *octree* we insert the objects sequentially and subdivide the spatial region into eight sub-regions, each of equal size.

  A termination criteria of splitting regions could be to have a maximum number of primitives in each leaf node.

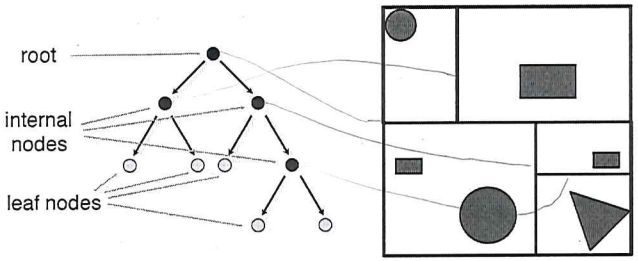


- KD-Trees: is a hierarchical spatial subdivision based on a *divide and conquer* approach.

  Step 1: Iteratively insert objects into the scene

  Step 2: Each time subdivide the cell where the object has been inserted if it was already occupied using axis aligned planes

The KD-tree can be represented as a tree data structure, where each non-leaf node contains pointers to its children. Leaf nodes are the regions containing the objects, they also store informations about their geometry and about the contained primitives.



Internal nodes store information about the separating axis aligned plane.

There exist different methods to decide how to split a region into its two subregions:

spatial median: a splitting plane is chosen and positioned at the median position of the node, along the longest dimension of the node. This means that at each splitting operation, the node is divided in two identically big children. The number of primitives contained in each node is not relevant in this algorithm. Note that in case of a non-uniform distribution of primitives, this method performs poorly.

primitive median: takes into account the number of primitives that will lie in each child node after the splitting, choose the splitting plane in order to divide the number of primitives equally among the two children.

SAH: surface area heuristics, assumes a uniform distribution of rays hitting the node and minimizes a splitting-cost to decide where to divide the node into its two children.

The splitting cost is defined as:

$$C = T + P_l N_l + P_r N_r$$

where T is the cost of traversing the node, P_l and P_r are the probabilities of hitting the left or right child respectively, N_l and N_r correspond to the number of primitives lying in each children after the subdivision. The probabilities of intersecting the left child (and respectively the right one) are given by:

$$P_l = \frac{SA(V_l)}{SA(V)}, \quad P_r = \frac{SA(V_r)}{SA(V)}$$

where $SA(V)$ corresponds to the surface area of voxel V.

SAH determines the splitting plane where the splitting cost is minimized.

The ray-region intersection test is performed as a *top-down* traversal: starting from the *root*, choose which children to examine.

- General BSP-Trees: Similar to *KD-trees*, but regions are split using arbitrary planes (no need for them to be axis aligned)

Bounding Volumes

An other way to accelerate ray-surface intersection queries is to *decompose objects* into overlapping *sets* bounded by simple volumes (spheres, AABB, OBB,..) in order to be able to perform fast rejections if the ray does not even intersect that volume.

The volumes containing the primitives are organized into a hierarchical tree structure in such a way, that smaller volumes are contained into larger volumes. A single large volume contains all the scene, and every ray-surface intersection query starts by this root volume. During the tree traversal, as soon as the ray does not intersect a volume region, the complete subtree contained by this volume is removed from the list of candidate intersections.

Bounding volumes containing primitives are stored as leafs in the hierarchical tree structure.

Bounding Volumes: The simple volumes one can choose to group the set of primitives can be of various types:

Sphere: Simples volume region, can be queried very fast for a ray-volume intersection test, suffers from empty regions inside: no tight fit

AABB: Axis Aligned Bounding Box

OBB: Oriented Bounding Box, harder to compute ray-OBB intersection test, but can pack primitives in a more compact way.

k-DOP: k Discrete Orientation Polytope, as OBB but with an arbitrary number of faces

others: Convex hulls, ...

As the bounding volume becomes more complex, the contained primitives are packed tighter and less ray-primitive intersection test will be performed. On the other side, a complex bounding volume is harder to query for an intersection test.

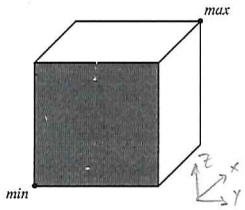
Also bounding volume hierarchies can be represented as a tree structure, where each non-leaf node has a pointer to its children:

Algorithm 7 Bounding Volume intersection test

```
1: function IntersectBVH(Ray r, Intersection& its)
2:     if boundingBox.hit(r) then
3:         if leaf then
4:             leaf.intersect(r, its)
5:         else
6:             leftChild.intersectBVH(r, its)
7:             rightChild.intersectBVH(r, its)
8:         end if
9:     end if
10: end function
```

Axis Aligned Bounding Box (AABB): defines a parallelepiped region of space aligned with the main axes, it is defined by two coordinates:

```
struct AABB
{
    Vector3f min;
    Vector3f max;
};
```



A ray-AABB intersection test is easy to perform and is computed as follows:

Step 1: find intersection with the yz planes (finding y and z coordinates of the intersection on the plane):

$$o_x + t_{x_1} d_x = x_{\text{min}} \qquad o_x + t_{x_2} d_x = x_{\text{max}}$$
$$t_{x_1} = \frac{x_{\text{min}} - o_x}{d_x} \qquad t_{x_2} = \frac{x_{\text{max}} - o_x}{d_x}$$

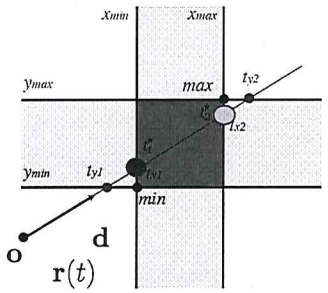
Step 2,3: find intersection with other planes finding $t_{y_1}, t_{y_2}, t_{z_1}, t_{z_2}$

Step 4: compute:

$$t_1^\star = \max\{t_{x_1}, t_{y_1}, t_{z_1}\}, \quad t_2^\star = \min\{t_{x_2}, t_{y_2}, t_{z_2}\}$$

Step 5: the ray intersects the AABB only if:

$$t_1^\star < t_2^\star$$



11 Participating Media (13p)

Until now we always assumed that the space between the objects in the scene is empty: vacuum. In reality the air filling the scene also interacts with light. *Participating media* need to be introduced in our renderings in order to simulate the presence of air (fog, smoke, ...) in the scene.

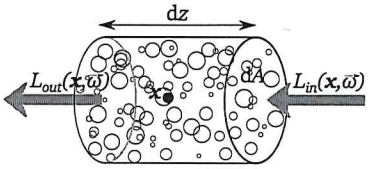
The introduction of a medium between any two surfaces has some consequences in the assumptions we made in the derivation of the rendering equation (7.1): the radiance does **not** remain constant along a segment of the light path, the incoming radiance $L_i(\mathbf{x}, \vec{\omega}_i)$ can not be replaced by the emitted/reflected radiance at an other position $\mathbf{x}' = r(\mathbf{x}, \vec{\omega}_i)$ in the scene:

$$L_i(\mathbf{x}, \vec{\omega}_i) \neq L_o(r(\mathbf{x}, \vec{\omega}_i), -\vec{\omega}_i) \tag{11.1}$$

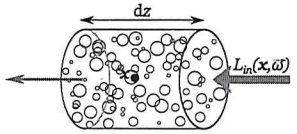
Physically speaking, the interaction between air and light is due to the presence of small particles suspended in the air. As we don't want to model those particles explicitly, we parametrize their properties with some coefficients.

Coefficients

We can model the interaction between a medium and radiance using some coefficients considering a *differential beam* containing a segment of a light path:

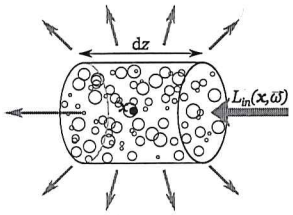


- **Absorption:** Due to the presence of particles in the air, some incoming radiance $L_{in}(\mathbf{x}, \vec{\omega})$ is absorbed. The amount of absorption is parametrized as follows:



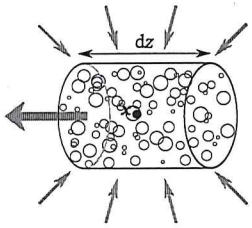
$$\mathrm{d}L(\mathbf{x}, \vec{\omega}) = -\sigma_a L(\mathbf{x}, \vec{\omega})\,\mathrm{d}z, \quad \left[\frac{1}{m}\right] \tag{11.2}$$

- **Out Scattering:** The floating particles in the air, absorb light but reflect it too. The reflected light is scattered in all directions and the intensity of the scattering is parametrized as follows:



$$\mathrm{d}L(\mathbf{x},\vec{\omega}) = -\sigma_s L(\mathbf{x},\vec{\omega})\,\mathrm{d}z, \quad \left[\frac{1}{m}\right] \tag{11.3}$$

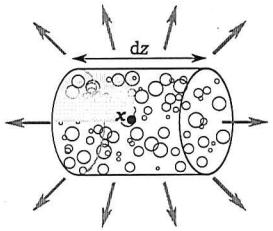
- **In Scattering:** In the same way as the *out scattering*, the suspended particles also reflect the incoming light (from all directions) toward the direction of the differential beam. The amount of in scattering is parametrized as follows:



$$\mathrm{d}L(\mathbf{x},\vec{\omega}) = \sigma_s L_s(\mathbf{x},\vec{\omega})\,\mathrm{d}z, \quad \left[\frac{1}{m}\right] \tag{11.4}$$

where $L_s(\mathbf{x},\vec{\omega})$ is the amount of in-scattered radiance (from all directions). Note that the in-scattering and out-scattering coefficients are the same.

- **Emission:** The floating particles may also emit light, the emission coefficient σ_a parametrizes this physical phenomenon:



$$\mathrm{d}L(\mathbf{x},\vec{\omega}) = \sigma_a L_e(\mathbf{x},\vec{\omega})\,\mathrm{d}z, \quad \left[\frac{1}{m}\right] \tag{11.5}$$

where $L_e(\mathbf{x},\vec{\omega})$ is the amount of emitted radiance (in all directions). Note that the emission and absorption coefficients are the same.

Radiative Transport Equation: combining all gains and losses to the intensity of radiance passing through a differential beam at position $\mathbf{x}$ in direction $\vec{\omega}$, we get:

$$\begin{aligned}\mathrm{d}L(\mathbf{x},\vec{\omega}) \;=\; & -\sigma_a L(\mathbf{x},\vec{\omega})\,\mathrm{d}z \quad -\sigma_s L(\mathbf{x},\vec{\omega})\,\mathrm{d}z \\ & +\sigma_a L_e(\mathbf{x},\vec{\omega})\,\mathrm{d}z \quad +\sigma_s L_s(\mathbf{x},\vec{\omega})\,\mathrm{d}z\end{aligned}$$

We can simplify the above equation introducing a new coefficient: the *extinction* coefficient $\sigma_t = \sigma_a + \sigma_s$, which is the sum of the absorption and scattering coefficient:

$$\begin{aligned}\mathrm{d}L(\mathbf{x},\vec{\omega}) \;=\; & -\sigma_t L(\mathbf{x},\vec{\omega})\,\mathrm{d}z \\ & +\sigma_a L_e(\mathbf{x},\vec{\omega})\,\mathrm{d}z \quad +\sigma_s L_e(\mathbf{x},\vec{\omega})\,\mathrm{d}z\end{aligned}$$

σ_t parametrizes the total *loss* of light per unit length.

Extinction along beam of finite length: ignoring in-scattering, emission and assuming a constant extinction coefficient σ_t, we can calculate the total loss of light intensity along a beam of finite length z as follows:

$$\begin{aligned}\mathrm{d}L(\mathbf{x},\vec{\omega}) &= -\sigma_t L(\mathbf{x},\vec{\omega})\,\mathrm{d}z \\ \int_0^z \frac{\mathrm{d}L(\mathbf{x},\vec{\omega})}{L(\mathbf{x},\vec{\omega})} &= \int_0^z -\sigma_t\,\mathrm{d}z\end{aligned}$$

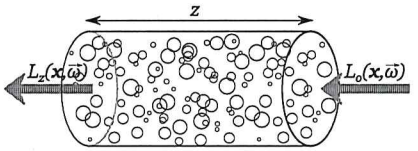
integrating both sides from 0 to z we obtain:

$$\ln L_z - \ln L_o = -\sigma_t z \Leftrightarrow \ln\frac{L_z}{L_o} = -\sigma_t z$$

exponentiating both sides we obtain the final result: the intensity of light reaching a point $\mathbf{y} = \mathbf{x} + z\omega$ is given by:

$$\frac{L_z}{L_o} = \overbrace{e^{-\sigma_t z}}^{\text{transmittance}} \Leftrightarrow L_z = L_o e^{-\sigma_t z} \tag{11.6}$$

The fraction of light that remains after traveling a finite distance through a medium with constant extinction coefficient is called *transmittance*.



Homogeneous VS Heterogeneous:depending on the nature of the medium, the transmittance $T_r(\mathbf{x},\mathbf{y})$ (ratio between radiance at $\mathbf{x}$ and at point $\mathbf{y}$) can be computed as follows:

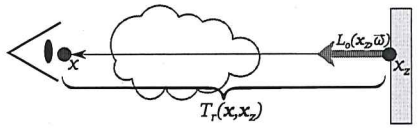
$$\text{Homogeneous: } T_r(\mathbf{x},\mathbf{y}) = e^{-\sigma_t\|\mathbf{x}-\mathbf{y}\|}, \quad \text{Heterogeneous: } T_r(\mathbf{x},\mathbf{y}) = e^{-\int_0^{\|\mathbf{x}-\mathbf{y}\|}\sigma_t(t)\,\mathrm{d}t}$$

Volume Rendering Equation

We can use the parametrized influence of a medium to the radiance traveling inside it to derive a new *rendering equation*.

- **Transmittance:** Consider the radiance reaching a point $\mathbf{x}$ after traveling through a distance $\|\mathbf{x} - \mathbf{x}_z\|$ in a homogeneous medium that cumulates a total transmittance of $T_r(\mathbf{x}, \mathbf{x}_z)$.

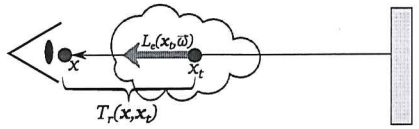
The losses of light when traveling through a medium are calculated with the *transmittance*, the reflected radiance at the intersection point $\mathbf{x}_z$ is reduced as:



$$L(\mathbf{x}, \vec{\omega}) = \boxed{T_r(\mathbf{x}, \mathbf{x}_z) L(\mathbf{x}_z, \vec{\omega})}$$

- **Emission:** Gains in radiance intensity are to take into account considering the total emission and total in-scattering.

The total emission is accumulated along the part of the ray-segment inside the medium and requires the evaluation of the integral of all infinitesimal (emissive) contributions at any position $\mathbf{x}_t$ on the ray between $\mathbf{x}$ and $\mathbf{x}_z$. Note that the emission coefficient may not be the same along the ray: $\sigma_a = \sigma_a(t)$

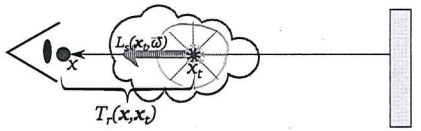


$$L(\mathbf{x}, \vec{\omega}) = T_r(\mathbf{x}, \mathbf{x}_z) L(\mathbf{x}_z, \vec{\omega}) + \boxed{\int_0^z T_r(\mathbf{x}, \mathbf{x}_t) \sigma_a(t) L_e(\mathbf{x}_t, \vec{\omega}) \, \mathrm{d}t}$$

- **In Scattering:** the inscattered radiance can be computed (ideally) in a similar way to the emissive contribution:

$$L(\mathbf{x}, \vec{\omega}) = T_r(\mathbf{x}, \mathbf{x}_z) L(\mathbf{x}_z, \vec{\omega}) + \int_0^z T_r(\mathbf{x}, \mathbf{x}_t) \sigma_a(t) L_e(\mathbf{x}_t, \vec{\omega}) \, \mathrm{d}t + \boxed{\int_0^t T_r(\mathbf{x}, \mathbf{x}_t) \sigma_s(t) L_s(\mathbf{x}_t, \vec{\omega}) \, \mathrm{d}t}$$

the problem consists in the evaluation of the total in scattered radiance $L_s(\mathbf{x}_t, \vec{\omega})$ for any position $\mathbf{x}_t$ along the ray as it requires an integration of the incoming radiance from every direction $\vec{\omega}'$ over a full sphere centered at $\mathbf{x}_t$:



$$\boxed{L_s(\mathbf{x}_t, \vec{\omega}) = \int_{S^2} f_p(\mathbf{x}_t, \vec{\omega}', \vec{\omega}) L_i(\mathbf{x}_t, \vec{\omega}') \, \mathrm{d}\vec{\omega}'} \quad (11.7)$$

where $f_p(\mathbf{x}_t, \vec{\omega}', \vec{\omega})$ is the *phase function* of the medium at point $\mathbf{x}_t$

Including this consideration, we can update the *volumetric rendering equation* and obtain:

$$\boxed{\begin{aligned} L(\mathbf{x}, \vec{\omega}) &= T_r(\mathbf{x}, \mathbf{x}_z) L(\mathbf{x}_z, \vec{\omega}) \\ &+ \int_0^z T_r(\mathbf{x}, \mathbf{x}_t) \sigma_a(t) L_e(\mathbf{x}_t, \vec{\omega}) \, \mathrm{d}t \\ &+ \int_0^z T_r(\mathbf{x}, \mathbf{x}_t) \sigma_s(t) \int_{S^2} f_p(\mathbf{x}_t, \vec{\omega}', \vec{\omega}) L_i(\mathbf{x}_t, \vec{\omega}') \, \mathrm{d}\vec{\omega}' \, \mathrm{d}t \end{aligned}} \quad (11.8)$$

Phase Function

Analogously to the surface **BDSF**, the *phase function* f_p describes the distribution of the scattered light.

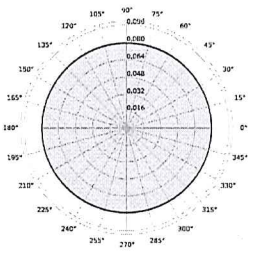
Unlike **BRDF**s, *phase functions* integrate to unity over the sphere, there is never energy dispersion:

$$\int_{S^2} f(\mathbf{x}, \vec{\omega}', \vec{\omega}) \, \mathrm{d}\vec{\omega}' = 1$$

Isotropic Phase Function: the simplest *phase function* one could imagine, is the one that scatters the incoming light $L_i(\mathbf{x}, \vec{\omega}')$ in all directions uniformly: the phase function is constant, and as it has to be normalized over the unit sphere, its value is:

$$\boxed{f_p(\mathbf{x}, \vec{\omega}', \vec{\omega}) = \frac{1}{4\pi}} \quad (11.9)$$

A visual representation of the isotropic phase function can be found drawing for each angle θ formed between $\vec{\omega}'$ and $\vec{\omega}$ the value of the phase function.



Anisotropic Phase Functions: we can introduce a measure of anisotropy g defined as:

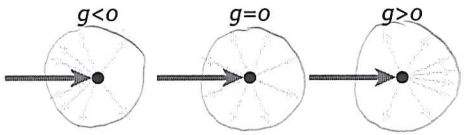
$$g = \int_{S^2} f_p(\mathbf{x}, \vec{\omega}', \vec{\omega})(-\vec{\omega} \cdot \vec{\omega}')\, \mathrm{d}\vec{\omega}'$$

The value of g determines the type of anisotropy:

$g = 0$ isotropic phase function

$g > 0$ forward scattering

$g < 0$ backward scattering



Henyey-Greenstein Phase Function: models anisotropic phase functions:

$$f_p^{HG}(\mathbf{x}, \vec{\omega}', \vec{\omega}) = \frac{1}{4\pi} \frac{1 - g^2}{(1 + g^2 - 2g\cos\theta)^{2/3}} \tag{11.10}$$

Changing the value of g, the *phase function* anisotropy varies: for $g \approx 1$ the phase function is almost a perfect forward scattering, where light continues to travel following the same direction. For values of $g \approx -1$ the phase function makes the medium behave like a surface that reflects all incoming light into the direction it came from.

Steam (vapor) has the property of forward scattering the incoming light, this is the reason why clouds appear to be white (light hits them on the top surface and light is scattered forward, following the direction they came from). On the other hand, *smoke* is backward scattering: light does not pass through it and is scattered backwards.

Solving Volumetric Rendering Equation - Analytically

The *Volumetric Rendering Equation* that needs to be solved to render scenes *filled* with a medium is the following:

$$L(\mathbf{x}, \vec{\omega}) = \underbrace{T_r(\mathbf{x}, \mathbf{x}_z) L(\mathbf{x}_z, \vec{\omega})}_{\text{absorbtion}} + \underbrace{\int_0^z T_r(\mathbf{x}, \mathbf{x}_t) \sigma_a(t) L_e(\mathbf{x}_t, \vec{\omega})\, \mathrm{d}t}_{\text{emission}} + \underbrace{\int_0^z T_r(\mathbf{x}, \mathbf{x}_t) \sigma_s(t) \int_{S^2} f_p(\mathbf{x}_t, \vec{\omega}', \vec{\omega}) L_i(\mathbf{x}_t, \vec{\omega}')\, \mathrm{d}\vec{\omega}'\, \mathrm{d}t}_{\text{in scattering}}$$

Implementing the *in scattering* component of the rendering equation is computationally expensive as for each point $\mathbf{x}_t$ lying on the ray we need to compute a spherical integral over the incoming light $L_i(\mathbf{x}_t, \vec{\omega}')$.

Single/Multiple Scattering: Note that the evaluation of the incoming radiance L_i may require other computations (path-tracing, ...). To avoid this complexity explosion we can simplify the process that happens in real life (the incoming light should be computed solving once again the *volumetric rendering equation*) we can assume that L_i arrives directly from a light source.

- **Single Scattering** assumes that $L_i(\mathbf{x}_t, \vec{\omega}')$ arrives directly from a light source:

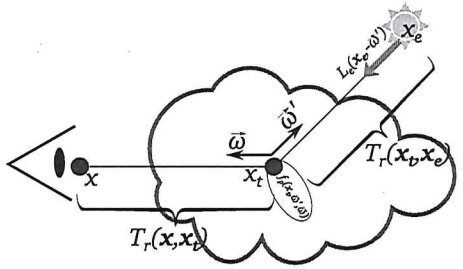
$$L_i(\mathbf{x}_t, \vec{\omega}') = T_r(\mathbf{x}_t, r(\mathbf{x}_t, \vec{\omega}'))\, L_e(r(\mathbf{x}_t, \vec{\omega}'), -\vec{\omega}') \tag{11.11}$$

- **Multiple Scattering** assumes that L_i arrives through multiple bounces (indirect illumination due as well to surface interaction with the scene, but also from scattering inside the medium).

Under the assumption of *single scattering* the in-scattering related part of the *Volumetric Rendering Equation* can be simplified:

$$L(\mathbf{x}, \vec{\omega}) = \ldots + \int_0^z T_r(\mathbf{x}, \mathbf{x}_t) \sigma_s(\mathbf{x}_t) \int_{S^2} f_p(\mathbf{x}_t, \vec{\omega}', \vec{\omega}) T_r(\mathbf{x}_t, \mathbf{x}_e) L_e(\mathbf{x}_e, -\vec{\omega}')\, \mathrm{d}\vec{\omega}'\, \mathrm{d}t \tag{11.12}$$

where $L_e(\mathbf{x}_e, -\vec{\omega}')$ is the light emitted from light source at position $\mathbf{x}_e$ in direction $-\vec{\omega}'$. The transmittance term $T_r(\mathbf{x}_t, \mathbf{x}_e)$ takes into account the possible loss of light intensity from the emitted radiance due to the presence of media in the segment $\overline{\mathbf{x}_t \mathbf{x}_e}$



Assuming a homogeneous medium, a point light as emitter, a simple phase function (isotropic) and no occlusion (ignore volumetric shadows), the evaluation of the *in scattering* term of the *volumetric rendering equation* can be performed analytically:

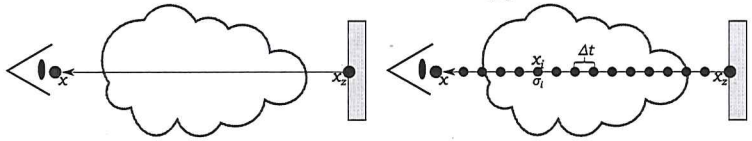
$$L(\mathbf{x}, \vec{\omega}) = \ldots + \underbrace{\frac{\Phi}{4\pi}}_{\text{emitted radiance}} \underbrace{\frac{1}{4\pi}}_{\text{phase function}} \underbrace{\sigma_s}_{\text{homogeneous medium}} \int_0^z e^{-\sigma_t \|\mathbf{x} - \mathbf{x}_t\|} e^{-\sigma_t \|\mathbf{x}_t - \mathbf{x}_e\|}\, \mathrm{d}t \tag{11.13}$$

Solving Volumetric Rendering Equation - Numerically

We can try to solve the *Volumetric Rendering Equation* numerically, i.e. approximating the integrals numerically.

Ray Marching: Discretize the integral computation in a sum (*Riemann Sum*):

$$L(\mathbf{x},\vec{\omega}) = \int_0^z T_r(\mathbf{x},\mathbf{x}_t)\sigma_s(\mathbf{x}_t)L_s(\mathbf{x}_t,\vec{\omega}) \qquad L(\mathbf{x},\vec{\omega}) \approx \sum_{i=1}^{N} T_r(\mathbf{x},\mathbf{x}_i)\sigma(\mathbf{x}_i)L_s(\mathbf{x}_i,\vec{\omega})\Delta t$$



The computation of the transmittance can be optimized as we know that the quantity is multiplicative:

$$T_r(\mathbf{x},\mathbf{z}) = T_r(\mathbf{x},\mathbf{y})T_r(\mathbf{y},\mathbf{z})$$

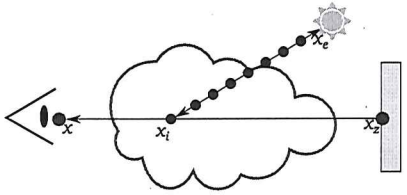
therefore, assuming constant extinction coefficient $\sigma_t(x)$ inside a discretization segment $\overline{\mathbf{x}_{i-1}\mathbf{x}_i}$ we have:

$$\boxed{T(\mathbf{x},\mathbf{x}_i) = T(\mathbf{x},\mathbf{x}_{i-1})e^{-\sigma_t(\mathbf{x}_i)\Delta t}}$$

The evaluation of the *in scattered* light L_s at every position $\mathbf{x}_i$ of the Riemann sum is still given by an integral equation (11.7) that can be discretized with Monte Carlo as follows:

$$\boxed{L_s(\mathbf{x},\vec{\omega}) = \int_{S^2} f_p(\mathbf{x}_t,\vec{\omega}',\vec{\omega})L_i(\mathbf{x}_t,\vec{\omega}')\,\mathrm{d}\vec{\omega}' \approx \frac{1}{M}\sum_{j=1}^{M}\frac{f_p(\mathbf{x}_i,\vec{\omega}'_j,\vec{\omega})L_i(\mathbf{x}_i,\vec{\omega}'_j)}{p(\vec{\omega}'_j)}} \qquad (11.14)$$

For single scattering evaluation, we assume $L_i(\mathbf{x}_i,\vec{\omega}'_j)$ arrives to $\mathbf{x}_i$ directly by an emitter located at $\mathbf{x}_e$. To calculate the value of the in-scattering radiance we have to perform ray-marching from $\mathbf{x}_i$ to $\mathbf{x}_e$.

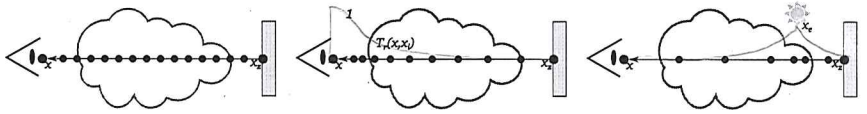


Ray Marching in Heterogeneous Media: distributing many points $\mathbf{x}_i$ uniformly along the camera ray may result ineffective as every point requires shooting one (or more) indirect lighting rays towards the light source, each of which needs to be ray marched as well to compute the transmittance $T_r(\mathbf{x}_i,\mathbf{x}_e)$.

We could distribute the position of those points $\mathbf{x}_i$ in a different fashion, i.e. considering the contribution that each of them will have to the final radiance intensity $L(\mathbf{x},\vec{\omega})$. Analyzing the behavior of the transmittance T_r as the point $\mathbf{x}_i$ is further away from the camera position $\mathbf{x}$ we observe that: the further the point is, the smaller its contribution to the total radiance intensity is.

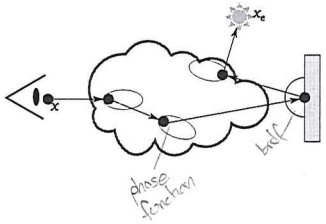
Starting from this observation we can position the samples $\mathbf{x}_i$ following a *pdf* which is proportional to the transmittance; doing so, we will position a lot of points $\mathbf{x}_i$ near the camera (with $T_r(\mathbf{x},\mathbf{x}_i)$ large) and only a few far away (where $T_r(\mathbf{x},\mathbf{x}_i)$ is small).

One could also distribute the points $\mathbf{x}_i$ according to an other part of the integral (for example the incoming radiance intensity).



$p(\mathbf{x}_i) = \text{const}$ $\qquad p(\mathbf{x}_i) \propto T_r(\mathbf{x},\mathbf{x}_i)$ $\qquad p(\mathbf{x}_i) \propto L_i(\mathbf{x}_i,\vec{\omega}') \propto \frac{1}{\|\mathbf{x}_i - \mathbf{x}_e\|}$

Volumetric Path Tracing



Following the same reasoning of the normal (surface) path tracing algorithm, if we would naively implement the idea of volumetric rendering we would fall into an explosion of complexity (each ray scatters M new indirect lighting rays for every one of the N points $\mathbf{x}_i$ lying on the camera ray).

Volumetric Path tracing reduces the exponential complexity of volumetric rendering as follows:

1) for each camera ray, sample a distance to next interaction (either volume or surface interaction)
2) scatter inside the volume (following the phase function) or bounce off the surface (following the **BRDF**).

Note that at each position $\mathbf{x}_i$ over the path inside the volume we have to choose a new direction $\vec{\omega}'$ to follow. This direction should be chosen according to the phase function f_p; the *pdf* of sampling a direction $\vec{\omega}'$ is equivalent to the value of the phase function for that direction.

The distance between each point of the path is also to be chosen. Ideally we would like to sample the distance according to the transmittance: in a thick medium (e.g. milk) we should choose short distances, while in a thin one (e.g. atmosphere) larger steps should be allowed:

$$\boxed{p(\mathbf{x}_{i+1}|\mathbf{x}_i,\vec{\omega}) \propto T_r(\mathbf{x}_i,\mathbf{x}_{i+1}) \Leftrightarrow p(t) \propto T_r(t)}$$

Sampling the distance in a homogeneous medium is simple as we have an analytical expression for the transmittance:

$$p(t) \propto T_r(t) \Rightarrow p(t) = \frac{e^{-\sigma_t t}}{\int_0^\infty e^{-\sigma_t t}\,dt} = \sigma_t e^{-\sigma_t t}$$

In order to sample a distance t according to the pdf, we have to compute the CFD:

$$P(t) = \int_0^t p(t')\,dt' = \int_0^t \sigma_t e^{-\sigma_t t'}\,dt' = 1 - e^{-\sigma_t t}$$

The inverted CFD (i.e. the sampling formula to have distances distributed proportionally to $T_r(t)$) reads:

$$\boxed{P^{-1}(\xi) = -\frac{\ln(1-\xi)}{\sigma_t}} \tag{11.15}$$

as expected, the sampled distance $t = P^{-1}(\xi)$ is inverse proportional to the thickness of the medium.

Algorithm 8 Volumetric Path Tracing - Homogeneous Volume

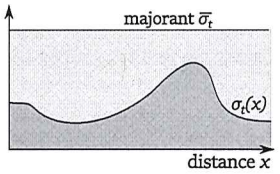
1: **function** VPT($\mathbf{x}$, $\vec{\omega}$)
2: $t = -\frac{\ln \xi}{\sigma_t}$ ▷ ξ and $1-\xi$ are both $\sim U(0,1)$
3: t_{max} =INTERSECT($\mathbf{x}, \vec{\omega}$) ▷ find nearest surface intersection
4: **if** $t < t_{max}$ **then**
5: $\mathbf{x} \mathrel{+}= t \cdot \vec{\omega}$
6: $\vec{\omega}' = PF$.SAMPLE($\vec{\omega}$) ▷ sample phase function
7: $pdf = \sigma_t e^{-\sigma_t t}$ ▷ probability of sampling t
8: **return** $\sigma_s \frac{T_r(t)}{pdf} \frac{PF.eval(\vec{\omega},\vec{\omega}')}{pdf_{PF}} \cdot$ VPT($\mathbf{x}, \vec{\omega}'$) ▷ $\frac{T_r(t)}{pdf} = \frac{e^{-\sigma_t t}}{\sigma_t e^{-\sigma_t t}} = \frac{1}{\sigma_t}$, $\frac{\sigma_s}{\sigma_t}$ is the albedo
9: **else** ▷ surface interaction
10: $\mathbf{x} \mathrel{+}= t_{max} \cdot \vec{\omega}'$
11: $\vec{\omega}' = BRDF$.SAMPLE($\vec{\omega}, \vec{\mathbf{n}}$) ▷ sample **BRDF**
12: $pdf = e^{-\sigma_t t_{max}}$ ▷ probability of hitting the surface
13: **return** $\frac{BRDF.eval(\vec{\omega},\vec{\omega}')}{pdf_{BRDF}} \frac{T_r(t_{max})}{pdf} \cdot$ VPT($\mathbf{x}, \vec{\omega}'$) ▷ $\frac{T_r(t_{max})}{pdf} = \frac{e^{-\sigma_t t_{max}}}{e^{-\sigma_t t_{max}}} = 1$
14: **end if**
15: **end function**

where at line 8 of algorithm 8 we have the property that the probability of sampling a direction $\vec{\omega}'$ with the phase function corresponds to the value of the phase function with that direction:

$$pdf_{PF}(\vec{\omega}') = PF.eval(\vec{\omega}, \vec{\omega}')$$

Woodcock Tracking

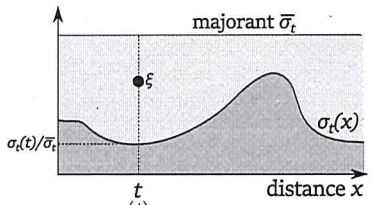
Woodcock tracking is a method to perform path tracing in a heterogeneous medium. The basic idea behind this method is the following:



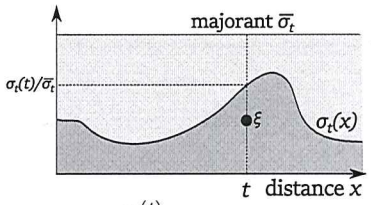
1) add a *fictitious volume* such that: real + *fictitious* volume is homogeneous (majorant extinction coefficients $\overline{\sigma}_t = \text{const}$)
2) generate free-path analytically (as for homogeneous volumes)
3) probabilistically accept/reject collisions based on local concentrations of real and *fictitious* volumes

Each time a new distance t is sampled as if the volume would be filled with a medium having $\overline{\sigma}_t$ as extinction coefficient, we need to perform a probabilistic approach to determine if the *collision* happened because of the original medium or because of the one that has been added to have have the homogeneous medium:

$$\text{real interaction:} \begin{cases} \text{yes} & \text{if } \xi < \frac{\sigma_t(t)}{\overline{\sigma}_t} \\ \text{no} & \text{else} \end{cases}$$



$\xi > \frac{\sigma_t(t)}{\overline{\sigma}_t} \Rightarrow$ no interaction



$\xi < \frac{\sigma_t(t)}{\overline{\sigma}_t} \Rightarrow$ interaction

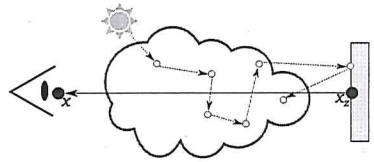
Note that the tighter the majorant $\overline{\sigma}_t$ is, the fewer sampled distances will be rejected.

Volumetric Photon Mapping

Can be used to have a good approximation of multiple scattering in volumetric rendering. Like the normal photon mapping, this is a two-pass algorithm:

Step 1: Trace photons from all light sources into the scene simulating their scattering.

Step 2: Trace rays from the camera and use photons to estimate multiple scattering



1. **Photon scattering:** the first phase of the volumetric photon mapping consists in scattering photons around the scene and to store them at different scattering positions. Note that those position can be on a surface but can also be inside a volume.

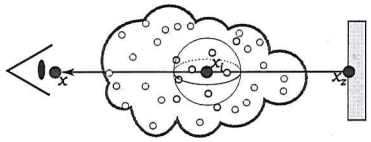
Algorithm 9 Volumetric Photon Scattering

```
function VolumetricScatter(x, ω⃗, Φ)
    t_max = Intersect(x, ω⃗)
    t = −ln ξ / σ_t                          ▷ free flight distance, assumed homogeneous volume
    if t < t_max then                        ▷ medium scattering
        x += t · ω⃗
        storeVolumePhoton(x, ω⃗, Φ)
        ω⃗' = PF.sample(ω⃗)
        VolumetricScatter(x, ω⃗', Φ σ_s/σ_t)
    else                                     ▷ surface interaction
        x += t_max · ω⃗
        storeSurfacePhoton(x, ω⃗, Φ)
        (ω⃗', pdf_BRDF) = BRDF.sample(ω⃗, n⃗)
        VolumetricScatter(x, ω⃗', Φ BRDF.eval(ω⃗, ω⃗') / pdf_BRDF)
    end if
end function
```

- **Volumetric Radiance Estimate:** when computing the volumetric rendering equation (11.8), specifically when evaluating the in-scattering radiance L_s:

$$L_s(\mathbf{x}_i, \vec{\omega}) = \int_{S^2} f_p(\mathbf{x}_i, \vec{\omega}', \vec{\omega}) T_r(\mathbf{x}_i, \mathbf{x}_e) L_e(\mathbf{x}_e, -\vec{\omega}') \, \mathrm{d}\vec{\omega}'$$

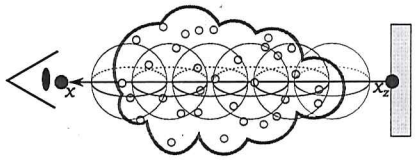
we can approximate this quantity considering the photons lying in a neighborhood of $\mathbf{x}_i$ as follows:



$$L_s(\mathbf{x}_i, \vec{\omega}) \approx \sum_{j=1}^{k} f_p(\mathbf{x}_i, \vec{\omega}_j, \vec{\omega}) \frac{\Phi_j}{\frac{4}{3}\pi r^3(\mathbf{x}_i)}$$

Merging this technique to approximate the in-scattering with the ray marching to approximate the integral along the ray we obtain the approximation to (11.12):

$$L(\mathbf{x}, \vec{\omega}) \approx \ldots + \sum_{i=1}^{N} T_r(\mathbf{x}, \mathbf{x}_i) \sigma_s(\mathbf{x}_i) \underbrace{\sum_{j=1}^{k} f_p(\mathbf{x}_i, \vec{\omega}_j, \vec{\omega}) \frac{\Phi_j}{\frac{4}{3}\pi r^3(\mathbf{x}_i)}}_{\approx L_s(\mathbf{x}_i, \vec{\omega})} \Delta t$$



Note that using ray marching combined with photon mapping, a lot of photons are counted multiple times, while others never give any contribution.

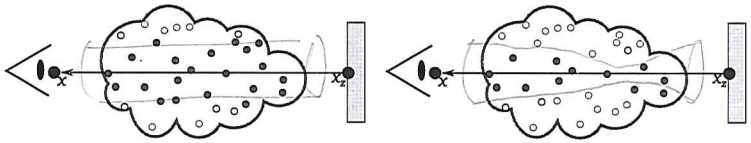
Beam Estimate: Instead of performing multiple neighbor queries and summing up all their contribution, the ideal would be to be able to find all important photons at once.

Enclosing all interesting photons inside a cylinder of fixed radius we can rewrite the approximation (11.12) as:

$$L(\mathbf{x}, \vec{\omega}) \approx \ldots + \sum_{i=1}^{N} T_r(\mathbf{x}, \mathbf{x}_i) \sigma_s(\mathbf{x}_i) f_p(\mathbf{x}_i, \vec{\omega}_i, \vec{\omega}) \frac{\Phi_i}{\pi r^2} \tag{11.16}$$

where $\mathbf{x}_i$ is the *orthogonal* projection of photon i on the camera ray $\mathbf{x} + t\vec{\omega}$.

Ideally one would prefer not to fix the cylinder radius, but to be free (for each distance t from the origin $\mathbf{x}$ of the ray) to choose a neighborhood radius.



Implementing efficiently a data structure storing photons that allows to perform variable radius beam queries is hard, but can be done.

The radiance estimate for a beam of variable radius is given by:

$$L(\mathbf{x}, \vec{\omega}) \approx \ldots + \sum_{i=1}^{N} T_r(\mathbf{x}, \mathbf{x}_i) \sigma_s(\mathbf{x}_i) f_p(\mathbf{x}_i, \vec{\omega}_i, \vec{\omega}) \frac{\Phi_i}{\pi r_i^2} \tag{11.17}$$

where r_i is the radius associated to the i-th photon.

Remark: the area over which we divide each photon power Φ_i is two dimensional for the beam estimate. In the traditional ray-marching each photon is blurred over a $3D$ sphere, while in beam estimation each photon is smeared only over a *disk* perpendicular to the camera ray.

The dimensionality reduction allows to have smaller *bias* due to the kernel approximation.

Photon Beams

It is possible to extend the concept of (point) photons to beams: entire ray segments which carry power Φ_i. The introduction of photon beams allows to have even better approximations for volumetric renderings.

12 Subsurface Scattering

There are materials that interact with light very particularly: they do not reflect light directly like diffuse materials, nor they absorb and scatter it like media.

We can model such materials (like ice, wax, marble, ...) introducing a new concept: the **BSSRDF** (**B**idirectional **S**cattering **S**urface **R**eflectance **D**istribution **F**unction), a property associated to *translucent* surfaces that determines the amount of light reflected at a given position $\mathbf{x}_o$ towards a given direction $\vec{\omega}_o$

$$f_r(\mathbf{x}, \vec{\omega}_i, \vec{\omega}_o) = \frac{\mathrm{d}L_o(\mathbf{x}, \vec{\omega}_o)}{L_i(\mathbf{x}, \vec{\omega}_i)\cos\theta_i \,\mathrm{d}\vec{\omega}_i} \qquad S(\mathbf{x}_i, \vec{\omega}_i, \mathbf{x}_o, \vec{\omega}_o) = \frac{\mathrm{d}L_o(\mathbf{x}_o, \vec{\omega}_o)}{L_i(\mathbf{x}_i, \vec{\omega}_i)\cos\theta_i \,\mathrm{d}\vec{\omega}_i}$$

To render surfaces having the property of subsurface scattering we need to solve the *generalized version of the reflection equation*:

$$L_o(\mathbf{x}, \vec{\omega}_o) = \int_A \int_{H^2} S(\mathbf{x}_i, \vec{\omega}_i, \mathbf{x}_o, \vec{\omega}_o) L_i(\mathbf{x}_i, \vec{\omega}_i) \cos\theta_i \,\mathrm{d}\vec{\omega}_i \mathrm{d}A(\mathbf{x}_i) \tag{12.1}$$

note that to calculate the reflected radiance at a point $\mathbf{x}_o$ we need to sum up all contributions from incoming light from every direction $\vec{\omega}_i$ at every point $\mathbf{x}_i$ over the surface A. For a **BRDF** surface, we had $\mathbf{x}_o = \mathbf{x}_i$, making the surface integration over A irrelevant, as it would have no contribution.

Diffusion Approximation

In highly scattering media ($\sigma_s \gg 1$) if one would compute equation (12.1) using standard methods (marching a ray inside the medium for each pair of points $\mathbf{x}_i$ and $\mathbf{x}_o$) one would need a lot of bounces for each light path, as the free flight distance t is very small for thick media.

Rendering this type of media requires a new method, as the conventional ones are too expensive.

A new model for rendering translucent materials finds its base in the following considerations:

1) the distribution of light in the translucent medium is modeled with the *diffusion approximation*, which describes the equilibrium distribution of illumination in highly scattering optically thick participating media
2) a solution to the diffusion equation is found by composing closed-form solutions of it for point light sources and an extension of a *dipole* of two light sources is used to approximate the overall scattering

Some assumptions are to be taken in order to use the model: *isotropic* scattering throughout the medium, the medium is *semi infinite*.

Principle of Similarity For an *anisotropic* scattering medium (where the phase function is not isotropic), with high albedo ($\sigma_s/\sigma_t \to 1$) the medium can instead be modeled as having a *isotropic* phase function with modified coefficients.

The *principle of similarity* is based on the observation that after many scattering events (that may occur inside a thick medium) the distribution of light inside the medium becomes more and more uniformly directionally distributed, regardless of the anisotropy of the phase function.

The reduced scattering coefficients are the following, where g is the anisotropy parameter:

Reduced scattering coefficient: $\sigma_s' = (1-g)\sigma_s$

Reduced extinction coefficient: $\sigma_t' = \sigma_a + \sigma_s'$

these new coefficients account for the effect of modeling the medium with an isotropic phase function approximation.

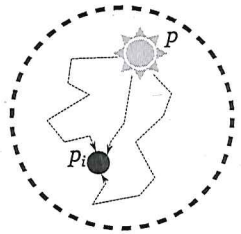
We can observe, that as $g \to 1$, the anisotropic phase function would allows a strong forward scattering of light, letting light continue its travel toward the same direction it had before the medium interaction. The value of the modified scattering coefficient σ_s' is almost zero, i.e. the isotropic simplification of the medium allows a free flight distance that is big, in other words allowing light to travel deeper inside the medium, having the same effect of a forward scattering phase function.

On the other hand, with $g \to -1$, the anisotropic phase function would almost reflect scattering light, each time in the opposite direction it came from: light makes almost no progress inside the medium. The value of the modified scattering coefficient σ_s' is high, emulating a thick, isotropic medium, approximating the effect of light having troubles making forward progress in a backward scattering medium.

Diffusion Approximation: In case of homogeneous, thick isotropic medium, we can solve the generalized reflection equation (12.1) (**BSSRDF**) analytically for an infinite medium, as we assume that the radiance that penetrates the Fresnel boundary, scatters only once at a depth of one mean free path $t = 1/\sigma_t'$. We can replace the scattering of incoming light with a virtual point light positioned at this depth, and solve the generalized reflection equation analytically.

The *fluence* $\Psi(\mathbf{p}_i) = \int_{S^2} L_i(\mathbf{p}_i, \vec{\omega}_i)\,\mathrm{d}\vec{\omega}_i$ at any point $\mathbf{p}_i$ due to a point light positioned at $\mathbf{p}$ can be computed as:

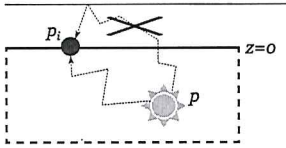
$$\Psi(\mathbf{p}_i, \mathbf{p}) = \frac{\Phi}{4\pi D} \frac{e^{-\sigma_{tr}\|\mathbf{p}_i - \mathbf{p}\|}}{\|\mathbf{p}_i - \mathbf{p}\|} \tag{12.2}$$



where D is the diffusion coefficient $D = 1/3\sigma_t'$ and $\sigma_{tr} = \sqrt{3\sigma_t'\sigma_a}$. Note that the fluence Ψ takes into account all incoming radiance L_i generated from the point light and scattering through the isotropic infinite medium ending in $\mathbf{p}_i$ after traveling through a random walk started at the emission position $\mathbf{p}$.

For subsurface scattering we need to analyze the case of a *semi infinite* medium, i.e. a medium that occupies all the space for $z \leq 0$. To solve the diffusion equation under this circumstances, a suitable boundary condition has to be found, such that we avoid to consider light paths that exit the medium and re-enter it after a few scattering events (which in reality do not happen, as if light leaves the medium, it continues in a straight direction without scattering anymore).

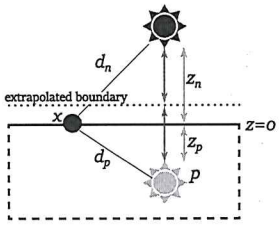
In particular we would like to be able to compute the fluence at the boundary of the medium ($z = 0$) as this boundary represents the surface of the translucent material; once the fluence has been computed, it is possible to find the corresponding value of the radiance.



We see that modeling the semi infinite medium naively does not work properly, as the solution (12.2) would take into account also light paths that traveled outside from the medium, which in the new settings should not exist.

To impose the boundary we need to introduce a *negative* light source by mirroring the existing one at an extrapolated boundary. The negative light source subtracts the light that leaves the medium, in this way we remove the excess of radiance reaching point $\mathbf{p}_i$ passing through an invalid path, i.e. through the upper half-space $z > 0$.

Introducing this negative light source (mirrored with respect to the extrapolation boundary) we have a zero-fluence at the extrapolated boundary. (some value subtracted)



The fluence at any given point $\mathbf{x}$ lying on the surface can by calculated as follows:

$$\Psi(\mathbf{x}) = \frac{\Phi}{4\pi D}\left(\frac{e^{-\sigma_{tr}d_p}}{d_p} - \frac{e^{-\sigma_{tr}d_n}}{d_n}\right)$$

From the fluence value we can estimate the one dimensional radial symmetric *dipole diffusion profile* $R(x)$ which describes the radial emittance:

$$R_d(\mathbf{x}) = \frac{\alpha'}{4\pi}\left(\frac{z_p(1+\sigma_{tr}d_p)e^{-\sigma_{tr}d_p}}{d_p^3} + \frac{z_n(1+\sigma_{tr}d_n)e^{-\sigma_{tr}d_n}}{d_n^3}\right) \tag{12.3}$$

where z_p corresponds to the depth where the first scattering will occur (mean free path distance) and z_n is chosen such that at the extrapolated boundary the fluence is zero:

$$z_p = \frac{1}{\sigma_t'}, \quad z_n = z_p(1 + \frac{4}{3}A)$$

Subsurface Scattering Algorithm

Now that we have a model to simulate subsurface scattering, we can implement an algorithm that will compute the subsurface scattered radiance in a efficient way. The algorithm is a *two*-pass algorithm:

Step 1: Sample Irradiance points on the translucent surface positioning random points on it and computing incoming radiance from all directions: irradiance caching.

Step 2: Evaluate Diffusion Dipole integrating over all cached irradiances and weighting them with the *reflectance profile* (12.3) (optimize this step introducing *hierarchical* evaluation, where distant points can be clustered)

13 Denoising

Instead of using a lot of samples per pixel in order to obtain a noise-free image, we can exploit *denoising* algorithms and apply them on (noisy) images obtained with a few samples per pixel.

Doing this allows to render an image much faster as we require a much lower number of samples, if the denoising algorithm works properly the resulting denoised image should be almost identical to the one rendered with a lot of samples.

Non Local Means Filter

Given a discrete noisy image I, the estimated value $V(i)$, for a pixel $i \in I$, is computed as a weighted average of all the pixels in the image:

$$V(i) = \sum_{j \in I} w(i,j)V(j)$$

where the family of weights $w(i,j)$ depend on the similarity between the pixels i and j. The weighting factors $w(i,j)$ satisfy the usual conditions:

$$0 \leq w(i,j) \leq 1, \quad \sum_{j \in I} w(i,j) = 1$$

The similarity between two pixels i and j depends on the similarity of $V(\mathcal{N}(i))$ and $V(\mathcal{N}(j))$, where $\mathcal{N}(k)$ denotes a square neighborhood of fixed size and centered at a pixel k.

The similarity between two regions $\mathcal{N}(i)$ and $\mathcal{N}(j)$ decreases exponentially accordingly to the *euclidean distance* of the two:

$$w(i,j) \propto e^{-\|V(\mathcal{N}(i)) - V(\mathcal{N}(j))\|^2}$$

A Posteriori VS A Priori

The following table shows the *pros* and *cons* for both, a *priori* and *posteriori* denoising methods:

	A Priori	A Posteriori
Performance	near optimal also for low *s.p.p.*	very effective for high *s.p.p.*
Flexibility	limited scope	general
Intrusivity	very intrusive: need for informations directly from the renderer as frequency informations, all samples to reconstruct light field	minor modifications, need for features buffers, variance informations

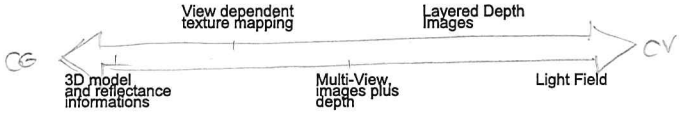
14 Image Based Rendering

Until now we rendered images starting from an accurate $3D$ description of a scene, where position, orientation, size and properties of each object were well specified. Light was calculated simulating its path with a physically based approach.

Some scenes are too difficult to be represented explicitly by their structure and properties: when their geometry (or scene lighting) becomes to complex, the traditional approach of simulating light in a physical manner fails.

Computer Vision

is the dual of computer graphics: we try to recreate a $3D$ model from a set of images. If combined with computer graphics, the model generated by the computer vision algorithm can be used to generate new images of the original scene (changing lighting, camera position, ...).



$3D$ model: classical rendering technique, where the scene is described by its $3D$ configuration and each surface property is explicitly modeled.

View dependent texture mapping: based on a $3D$ model and textures applied to the surfaces. The textures are extracted from real images, and are warped to the $3D$ model surface. When rendering the model, the textures (coming from different images) are blended together depending on their camera pose, distance, ...

Multi-View: in addition to the color information for each pixel, a special camera registers also the depth information. Virtual cameras can be moved in the scene and the image can be reconstructed querying the original color values plus the depth at each pixel.

It is trivial that a lot of pixels seen by the virtual camera have no correspondent point in the original image. This fact causes regions where the pixel intensity of the virtual camera can not be calculated (the mapping between the original image I and the virtual one I_{virt} is not *surjective*).

This problem can indeed be fixed registering multiple color-depth images with different perspectives. During the reconstruction process (generation of I_{virt}) a interpolation happens between the different sources.

Light Field

Assuming that we can measure, in a scene, everything that happens with light, i.e. for we can measure the light intensity for each of the following dimensions:

θ: the first coordinate of a light ray

ϕ: the second coordinate of a light ray

x, y, z: the origin of the light ray

λ: the ray wavelength distribution (ray color)

t: moment in time where the measurement happens

all light traveling in every direction

given a scene-specific *plenoptic* function

$$P_{scene}(\theta, \phi, x, y, z, \lambda, t) \tag{14.1}$$

we would be able to render the scene from any virtual position of the camera (selecting the ray origin x, y, z corresponding to the camera position).

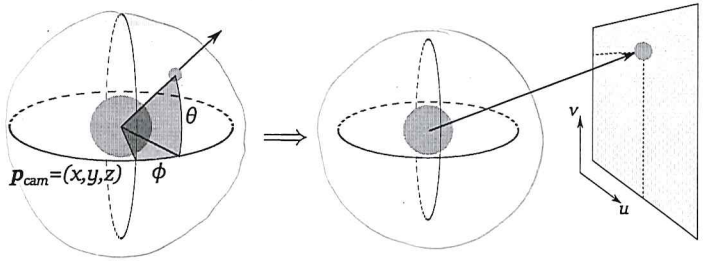
As we want to sample the RGB color of each single pixel of what the virtual camera sees at a given instant t, the plenoptic function can be reduced from 7-dimensional to 5-dimensional:

$$P(\theta, \phi, x, y, z, \lambda, t) \Rightarrow P_{\lambda,t}(\theta, \phi, x, y, z)$$

once we fix the camera position $\mathbf{p}_{cam} = (x, y, z)$, the resulting image $I_{\mathbf{p}}$ is given by sampling the plenoptic function over the θ and ϕ direction.

Furthermore, note that we do not need to parametrize a ray by 5 dimensions, knowing that a ray is infinite in space, its origin can be moved along its direction without changing the ray itself. We can reduce its description from $5D$ to $4D$

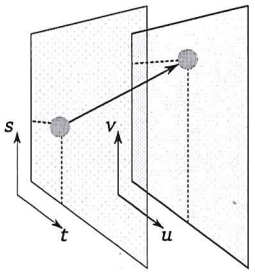
Introducing an *image plane*, the spherical coordinates can be mapped to their corresponding coordinates on the plane.



Lumigraph: The plenoptic function has been simplified for our purposes of rendering a static image and is now only 4-dimensional. The $4D$ plenoptic function is called *lumigraph.*

A change of variables will allow to define the lumigraph with help of two planes: the image plane uv and the camera plane st:

$$P(u,v,x,y,z) \Rightarrow \boxed{P(u,v,s,t)}$$



A virtual image I_{virt} can be computed sampling the lumigraph keeping s,t constant and varying u,v according to the image plane dimension.
Changing also s and t will result into a virtual image taken from a different position.

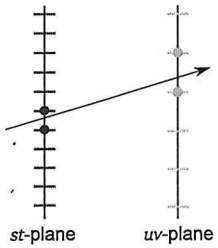
If we have a collection of images of the same object, but taken from different camera positions, this would correspond to a discrete sparse sampling of the lumigraph: for some points (s,t) we have a complete description of the uv domain of the lumigraph.

Such a collection of images can be acquired using a camera array, where multiple sensors are placed in different positions. This representation of the lumigraph is called *Field radiance - uv array of st images.*

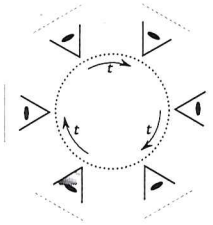
The dual of this representation of the lumigraph, happens when keeping u and v constant (on the image plane) but varying the camera position. This representation of the lumigraph is called *Surface radiance - st array of uv images.*

Light Field Rendering: once we have acquired the lumigraph of an object, we can render a scene representing it from the perspective of a virtual camera. This is done shooting a camera ray, determining the u,v,s,t coordinates of the lumigraph that correspond to the camera ray and finding the RGB value of the pixel by interpolating near values of the lumigraph.

Note that for a linear interpolation we need to interpolate 16 values of the lumigraph, as we are in $4D$: quadrilinear interpolation.



- $3D$ **Light Fields:** keeping only one parameter constant and varying the other three we can generate a $3D$ subspace of the light field.



Fixing s and varying t, sample for each position t the complete uv plane. The union of all sampled images along the t direction of the st plane corresponds to a set of images, all of the same object but taken from different horizontal positions.

If we change the geometry of the st plane, making it a cylinder (or a sphere), when moving along the t direction we perform a circle *looking* outwards.

This representation of a $3D$ slice of the lumigraph is called *concentric mosaic,* and once we have them, we can reconstruct any view of the scene from a virtual camera placed inside the circle t.

Advantages and Cons of Light Fields and other methods

The following chart illustrates the main advantages and disadvantages of using light field instead of using classical rendering techniques based on ray-tracing and $3D$ models:

Advantages	Disadvantages
WYSIWYG	WYSIAYG
simpler computations	static geometry
cost *independent* of the scene complexity	fixed lighting
cost *independent* of material properties	high storage costs

here a comparison with other rendering techniques:

	Acquisition	Reconstruction	Rendering
Classical	few cameras (if any)	full $3D$ geometry and reflectance reconstruction, difficult for complex scenes, often applied for specific objects	classical computer graphics
Depth-based	few cameras	depth estimation error prone	depth-based view interpolation
Image-based	dense sampling, many cameras	lumigraph requires a lot of storage	view interpolation, light field rendering

Index

—$\mathcal{A}$—
AABB, 45
  intersection, 45
absorbtion, 46
albedo, 9, 59
Archimede's Hat-Box therem, 15

—$\mathcal{B}$—
backward scattering, 51
barycentric coordinates, 4
Beer Lambert law, 48
blackbody, 6
bounding volumes, 44
  AABB, 44
  k-DOP, 44
  OBB, 44
  sphere, 44
BRDF, 8
  anisotropic, 8
  diffuse, 8
  Helmholtz reprocipity, 8
  isotropic, 8
  microfacet, 12
  reflection, 9
  refraction, 10
    index of refraction η, 10
  refraction VS reflection, 11
  symmetry, 8
BSSRDF, 59
BTDF, 11

—$\mathcal{C}$—
camera
  pinhole, 27
  thin lens, 27
concentric mosaic, 65
cylinder
  sampling, 15

—$\mathcal{D}$—
depth of field, 27
diffusion
  approximation, 59
  equation, 59
    diffusion coefficient, 60
dipole, 59

—$\mathcal{E}$—
emission, 47
emitters
  environment light, 25
  mesh light, 21
  point light, 21
  sphere light, 21
emitters:VPL, 33
eucludean distance, 62
extinction coefficient, 48

—$\mathcal{F}$—
Fermat's principle, 9
filter, 62
fluence, 60
forward scattering, 51
Fresnel law, 11

—$\mathcal{G}$—
geometry term, 19, 33

—$\mathcal{H}$—
HG phase function, 51
hierarchichal structure, 44

—$\mathcal{I}$—
illumination map, 35
inversion method, 13
  $2D$ distributions, 14
  CFD, 13
  transformations, 13, 19
    jacobian, 20

—$\mathcal{K}$—
kernel, 38
  cone, 38
  gaussian, 38
  uniform, 38

—$\mathcal{L}$—
lambertian surface, 8
light
  bioluminescence, 6
  chemiluminescence, 6
  fluorescence, 6
  incandescence, 6
  luminescence, 6.
  phosphorescence, 6
light field, 64
light path, 63
  caustic, 28, 32, 35
  direct illumination, 28
  Heckbert's classification, 28
  indirect illumination, 28
lumigraph, 65
  interpolation, 65

—$\mathcal{M}$—
many light rendering, 33
  VPL, 33
microfacet, 12
  Beckmann distribution, 12
MIS, 22
  fireflies, 22
  variance, 22

—$\mathcal{N}$—
non local mean filter, 62

—$\mathcal{P}$—
participating media, 46
  absorbtion, 46
  Beer Lambert law, 48
  emission, 47
  extinction, 48
  homogeneous VS heterogeneous, 48
  in scattering, 47
  out scattering, 47
  reduced coefficients, 60
path tracing
  volumetric, 54
phase function, 50
  HG, 51
  anisotropic, 51
  isotropic, 50
  sample, 54
photon beams, 58
photon mapping
  algorithm, 36
  emission, 36
  indirect light, 37
  kernel, 38
  photon storage, 37
  progressive photon mapping, 40
  relaxation, 41
  volumetric, 56
Plank's law, 6
plenoptic function, 64
principle of similarity, 59

—$\mathcal{Q}$—
quadrilinear interpolation, 65

—$\mathcal{R}$—
radiance
  fluence, 60
  symmetry, 7, 29, 46
radiometry, 6
  flux, 6
  irradiance, 6
  radiace, 7
  radiant intensity, 7
  radiosity, 7
ray equation, 3
ray marching, 53
ray tracing
  backward, 3, 8, 18
  bidirectional, 18, 32
  forward, 3, 8, 18
  path tracing, 30
    russian roulette, 30
  ray-surface intersection, 42
  recursive MC, 29
  double counting, 30
  pseudocode, 29
reduced coefficients, 60
  extinction, 60
  scattering, 60
reflection equation, 8, 9, 18
  area, 19
    geometry term, 19, 33
    visibility term, 19
  generalized, 59
  hemisphere, 19
Riemann Sum, 53
russian roulette, 30, 37

—$\mathcal{S}$—
sampling
  cylinder, 15
  discrete pdf, 26
  hemisphere
    cosine-weighted, 16, 22
    uniform, 16
  sphere, 15, 21
  spherical cap, 21
  strategy
    latin hypercube, 17
    multi jittered, 17
    random, 17
    stratification, 17
scattering
  backward, 51
  forward, 51
  in, 47
  multiple, 56
  out, 47
  single VS multiple, 51
Snell's law, 10, 11
spatial subdivision, 42
  bsp-tree, 44
  kd-tree, 43
  octree, 43
sphere
  bounding volume, 44
  sampling, 15, 21
subsurface scattering, 59
  algorithm, 61

—$\mathcal{T}$—
translucent, 59
transmittance, 49
tree structure, 44

—$\mathcal{V}$—
virtual point light, 33
volumetric path tracing, 54
volumetric photon mapping, 56
  beam estimate, 58
  spherical estimate, 57
voxel, 42

—$\mathcal{W}$—
Woodcock tracking, 56